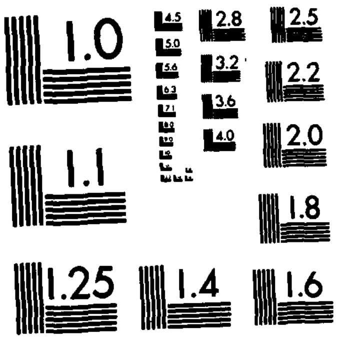

1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

ADA124821

DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT DIGITAL ENGINEERING LABORATORY

THESIS

AFIT/GCS/EE/82D-21 Eric F. Imker
Captain USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

83 02 022 211

AFIT/GCS/EE/82D-21

DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT DIGITAL ENGINEERING LABORATORY

THESIS

AFIT/GCS/EE/82D-21 Eric F. Imker
Captain USAF

AFIT/GCS/EE/82D-21

DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT DIGITAL ENGINEERING LABORATORY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Eric F. Imker, B.S.
Captain USAF
Graduate Computer Science

December 1982

Preface

The thought of a distributed data base is an appealing concept for practical solutions to real life problems. For instance, an inventory of parts available for a specific time period may be stored on one computer. A list of the people who will use them during the same time period might be stored on another computer. Connecting these two data bases together would provide an excellent purchase order strategy for management of the parts inventory. It seems like such a simple thing to do, until you examine the technical requirements involved. This thesis examines the issues in a distributed data base system and attempts to offer a design for implementing one in the Digital Engineering Laboratory at AFIT.

In researching this topic I discovered that the terminology can be very confusing. Keywords in one source mean different things in another source. Once you have identified the similiar concepts in different sources, you discover that distributed data base issues are well defined. The solutions to the problems raised by these issues is another story.

I wish to thank Dr. Thomas C. Hartrum for his assistance as my thesis advisor. I would also like to thank Major Walter D. Seward and Dr. Gary B. Lamont for their contributions as thesis committee members. Finally, my wife, Debbie deserves a honorary masters degree in epistemology for her support and encouragement over the last 18 months.

Contents

List of Figures

List of Tables

Abstract

A distributed data base management system (DDBMS) was designed with the goal of providing the AFIT School of Engineering a research tool. The objective was to use state-of-the-art knowledge in a design that would provide an experimental testbed to further advance DDBMS knowledge.

Toward this goal, an extensive investigation was made into distributed data base systems. Numerous alternatives were presented in the areas of configurations and architecture, data allocation, directory management, query processing, concurrency control, reliability, integrity and security. This background discussion includes advantages and disadvantages of the alternatives.

A top level system design was presented which includes replication of data, a universal data model and query language, a centralized and extended centralized directory system and the majority voting concurrency control algorithm. Due to the complexity and size of the development effort, a detailed design of only the directory system was made in this research effort. Follow-on development efforts over the next few years will be required to complete this research project.

DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT DIGITAL ENGINEERING LABORATORY

I. INTRODUCTION

Background

The purpose of any data base system is to record information and be able to provide a user easy access to this information. Most organizations use data bases that are on a single computer to make management decisions or to service customers. This is typically known as a centralized data base system. As more data is required to be stored and more users require access to the data, larger, more sophisticated computers are needed. Eventually a centralized data base system becomes undesirable for two reasons. First, data retrieveal slows to an unacceptable speed and becomes inefficient. Second, the cost factor of a very large computer becomes prohibitive.

The alternative to the single computer is a network of independent computers sharing the data. The need to distribute data bases throughout a computer network has come not only from hardware limitations, but also from the need to locate the data bases over large geographic areas.

When one describes such a network it is usually called a "computer network". However, the actual data exchange and how it is accomplished are characteristics which place different computer networks into different catagories. One network might simply exchange whole files of data or transmit an entire computer program when one computer requests it. Another network might have commands being exchanged between computers which require processing to be accomplished at one computer and then only the results of the processing are transmitted. The key to the network philosophy is that data is stored at different physical locations. In other words, data is distributed over the network. When commands are exchanged to request only the result of processed data, it is called either a distributed processing system or a distributed data base system depending on which definition you wish to use (Ref 8:1-3).

There are basically two types of physical configurations used in computer networks. They are the "star" and the "network" configurations. The star configuration is used when one computer acts as a central controller. A disadvantage to this concept is that inefficient operation of the star may occur because the central controller cannot forward commands fast enough (Ref 4:69-72). The "network" configuration is used to link all the individual computers to each other directly or some subset thereof.

Once the physical computer network is established and communication problems are solved, consideration must be given to controlling the network.

Issues such as:

(1) Language selection for data definition and manipulation;

(2) Logical and physical data base design;

(3) Application programming;

(4) Query processing;

(5) Concurrency control;

(6) Reliability;

(7) Integrity;

(8) Security

must be addressed. Additionally, other problems such as where to store data and how to locate data must be addressed in distributed data base systems. Research is continuing for new solutions in all these areas (Ref 2:1).

Statement of the Problem

The Air Force Institute of Technology (AFIT) Digital Engineering Laboratory (DEL) is currently involved in several areas of data base research. A distributed data base system sharing stored data would provide a significant upgrade of the existing facility. In a few years, memory limitations would be reached on individual computer systems. At that time an operational distributed data base would be valuable. This upgrade would be intended initially as a research tool for the DEL. Eventually the distributed data base would become a pedagogical tool for students here at AFIT as well as an operational tool.

Scope

This thesis effort was concerned with the design of a distributed data base management system (DDBMS) for the DEL. The specific purpose of this thesis was threefold: first, to investigate current distributed data base technology; second, to establish long term and short term requirements and objectives for a DDBMS in the DEL; and third, to provide a top level design of a DDBMS system that meets the requirements of the DEL. Desirable alternatives of the issues as discussed in Chapter 2, Distributed Data Base Review, were provided for in the design. It was obvious from the beginning that a detailed design of a complete DDBMS was impossible in the time period available for this investigation. However, a design of the directory system was completed.

Approach

An extensive literature search was made to analyze operational distributed data base management systems (DDBMS). The objective here was to avoid a "reinvent the wheel" type of approach. Unfortunately, most operational DDBMSs are designed for a unique environment; however, basic concepts are the same. The underlying purpose of this research was to establish long term requirements and objectives for a DDBMS in the DEL which could eventually be implemented. After an overall long term lab plan was developed, a short term plan was established. Following this, a top level system design for the short term DDBMS plan was made. Finally, design of the directory management system component of the short term plan was made. This effort used software engineering tools with the intent that follow-on thesis efforts could provide the remaining design plan in preparation for implementation with modifications as needed.

Overview of the Thesis

The format of this thesis basically follows the approach taken during the research. Chapter II presents a background review of distributed data base concepts. It presents alternatives of key issues including advantages and disadvantages of each. Chapter III presents long term and short term requirements for a distributed data base management system in the Digital Engineering Laboratory. Chapter IV presents the top level system design for the DDBMS to be implemented. Chapter V describes the directory management system component of the design in more detail. Chapter VI presents a summary of this investigation and recommendations for the future.

II. DISTRIBUTED DATA BASE REVIEW

Introduction

This chapter presents a background review of distributed data base concepts. It is divided into eight sections as follows:

(1) Configurations and architectures;

(2) Data allocation;

(3) Directory management;

(4) Query processing;

(5) Concurrency control including deadlock issues;

(6) Reliability;

(7) Integrity;

(8) Security.

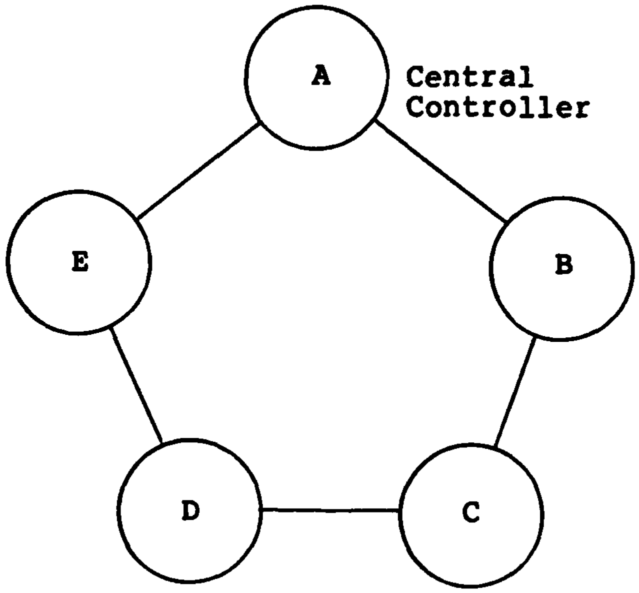
The order in which these sections are presented does not reflect any degree of importance. These issues are all related to each other and vary in importance with the individual network being discussed. Numerous articles have been devoted to each of these issues. This chapter will summerize the issues and provide a quick reference for material presented later in this thesis.

Configurations and Architectures

When discussing computer configurations, one must first identify the logical and physical communication links. The physical configurations available in a distributed data base (DDB) system are similiar to any other computer network. They are the star, ring, linear, fully connected, hierarchical and network configurations. (The term network is an unfortunate word in this sense. The term "network configuration" here implies a general topological interconnection among computers. A "computer network" refers to a communication subsystem.) The physical connections may or may not match the logical communication routes. The logical communication is the key factor in a computer network. For instance, a logical star configuration implies that all messages are sent to a central controller for processing and distribution. However, the physical communications might be interconnected in a loop. Figure 1 illustrates this example with a general data exchange, not necessarily the messages in a DDB system. If the network exists primarily to serve a DDB management system (DDBMS), then the physical and logical configuration should match if possible (Ref 5:48,164).

In a DDB system the data base spans the entire network. The partitioning of the DDBMS functions between the computers identifies the specfic DDB architecture. There are three different strategies for approaching the DDB management architecture. They are integrated, homogenous, and heterogenous (Ref 19:351-357). In discussing DDB architectures the physical communication channels will not be of any concern.

In the integrated architecture approach a portion of



Step	Action
1	Computer C addresses a message to the central controller to process data found in computer E.
2	The message is routed through computer B to the central controller.
3	Central controller addresses a command to computer E to process the data.
4	Computer E processes the data.
5	Computer E transmits the results of the processed data back to the central controller.
6	Central controller forwards the results to computer C through computer B.

Figure 1 Operation of a Logical Star Configuration in a Physical Loop Configuration

processing capability of each computer is dedicated to network data exchange. Each individual data base management system (DBMS) knows about the others and has access to them without data translation between the computers (Figure 2a). Unfortunately this strategy reduces useful CPU time at each computer and requires memory to store the data exchange processing technique.

In the homogenous approach the function of network data exchange is removed from each computer and a "communication box" is attached between each computer and the rest of the network (Figure 2b). Each computer must support the same DBMS as the others. This solution has become a standard method for implementing DDB networks.

The heterogenous approach is used when computers with different DBMSs (storing techniques) need to be linked together. For instance, one computer might store data using a relational approach (table structure). Another computer might use a hierarchical approach (tree structure) (Ref 7: 63-80). Since the data bases are incompatable, a "translator box" is inserted between the "communication box" and the communication channel, as in Figure 2c. If a universal language is developed to which each DBMS can translate, then the "translator box" is inserted between the computer and the "communication box". The heterogenous strategy is also a popular method for implementing DDB networks. However, it is a much more difficult system to implement (Ref 12).

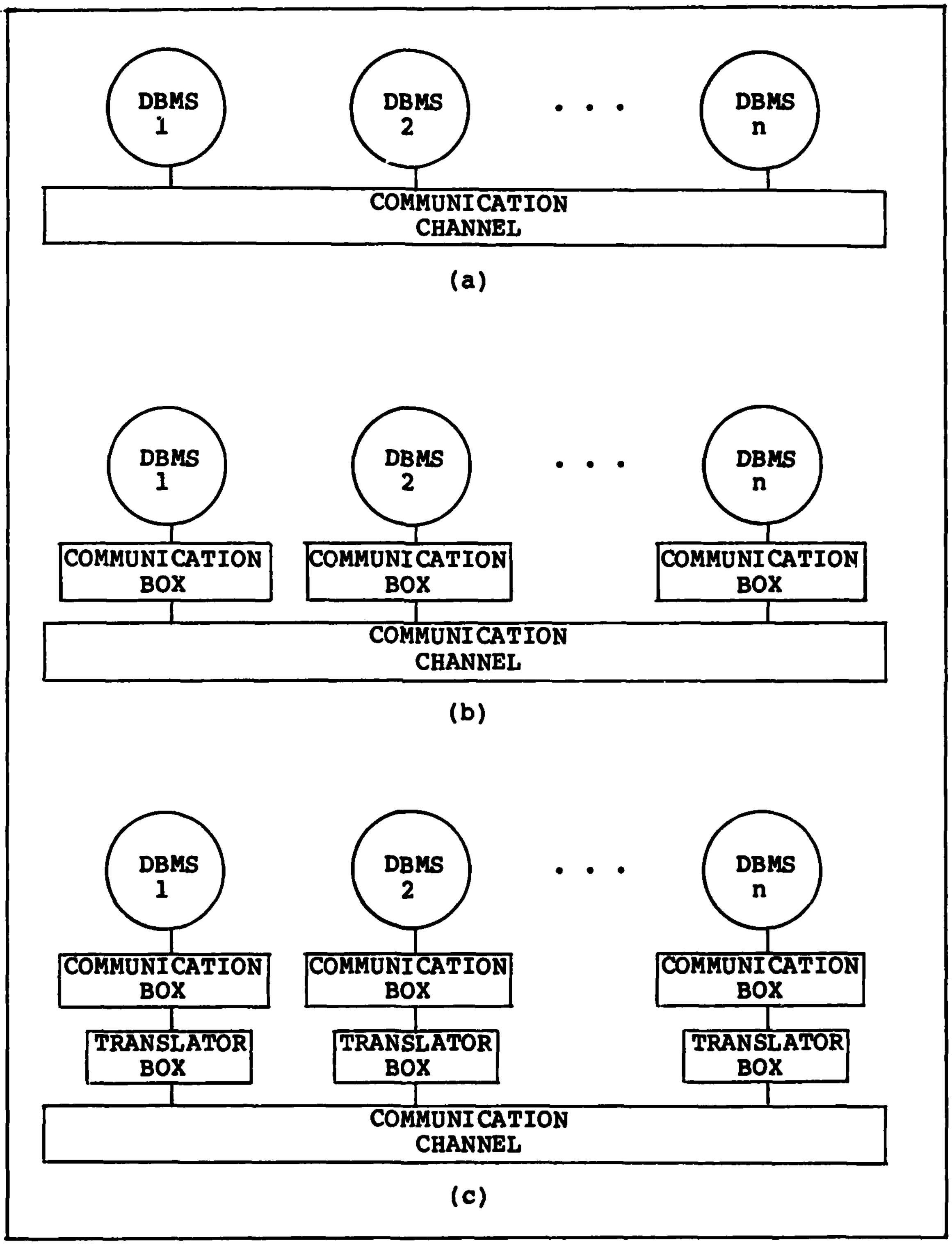


Figure 2 (a) Strategy One, Integrated Architecture
(b) Strategy Two, Homogenous Architecture
(c) Strategy Three, Heterogenous Architecture
(Ref 19:352)

Data Allocation

The question of where to store data in a DDB system is significant. A DDB system implies that data will be stored at different computers. The objective then is to reduce the communications required to move the data thereby increasing speed and reducing communication costs. An optimum solution for storing data at particuliar computers is a complex problem. One might first consider storing data at the computer where it is most frequently used. However, other factors should be considered such as:

(1) Nonlocal data accesses;

(2) Data distribution methods;

(3) Update frequency;

(4) Number of computers;

(5) Amount of data;

(6) Communication cost and speed.

A nonlocal data access is one that can not be handled locally within a DBMS's own data base. In this case, a system directory must be used to transfer a request for data to a remote computer. This is called an "exception" or a "miss" at the local computer. The remote computer that receives a data request calls that access an alien access. The number of alien accesses during a given time is called the specificity. A local data base that receives a high number of alien accesses is said to be highly specific (Ref 5:48-52).

The data distribution method has a wide range of possibilities. In general, the three basic ways exist to distribute data at different computers are to centralize, partition, or replicate.

A centralized scheme describes a data base at one computer which contains all data ever used by all the other computers. This definition does not restrict the other computers from duplicating a portion of it for themselves. The obvious advantage is that no misses ever occur at the centralized site. Also, data that is very rarely used by other computers can be stored at the centralized data base. There are three main disadvantages to centralizing in a distributed data base environment. First, more than one remote computer may be asking for access to the centralized data base. Second, the response time is usually slow because of the centralized data base size. Third, a failure at the centralized site means a significant degradation to the distributed data base network.

A partitioned data base occurs when the entire data base is divided into disjoint sets. Each computer is assigned a portion which it uses the most (Figure 3). In this situation the advantage is that storage costs are minimized because of no duplication of data. The main disadvantage is that more misses are likely to occur, thereby increasing communication costs.

A replicated data base exists when the same data is found at two or more sites in the system, as in Figure 4.

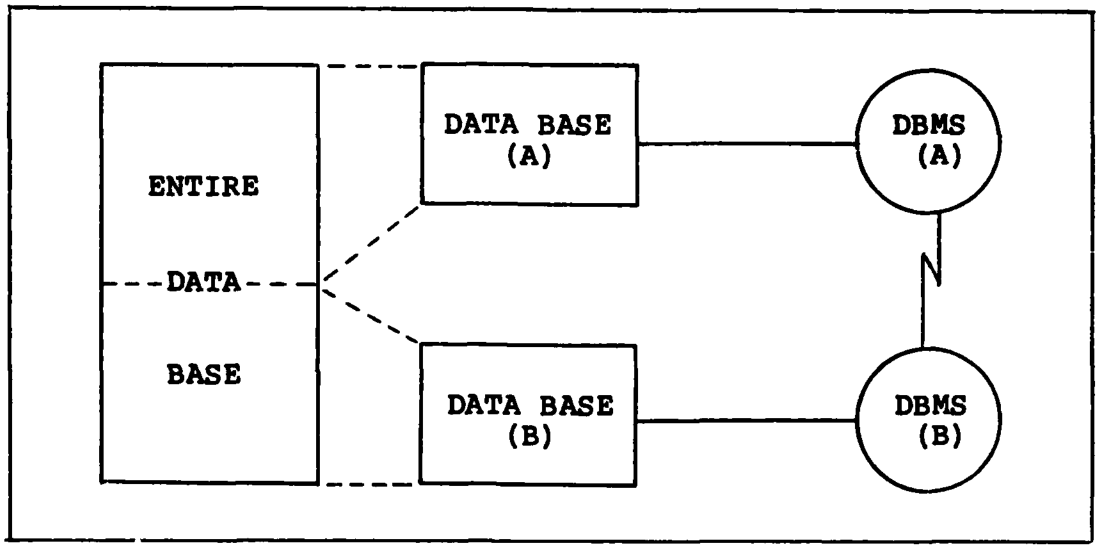


Figure 3 Partitioned Data Base (Ref 5:49)

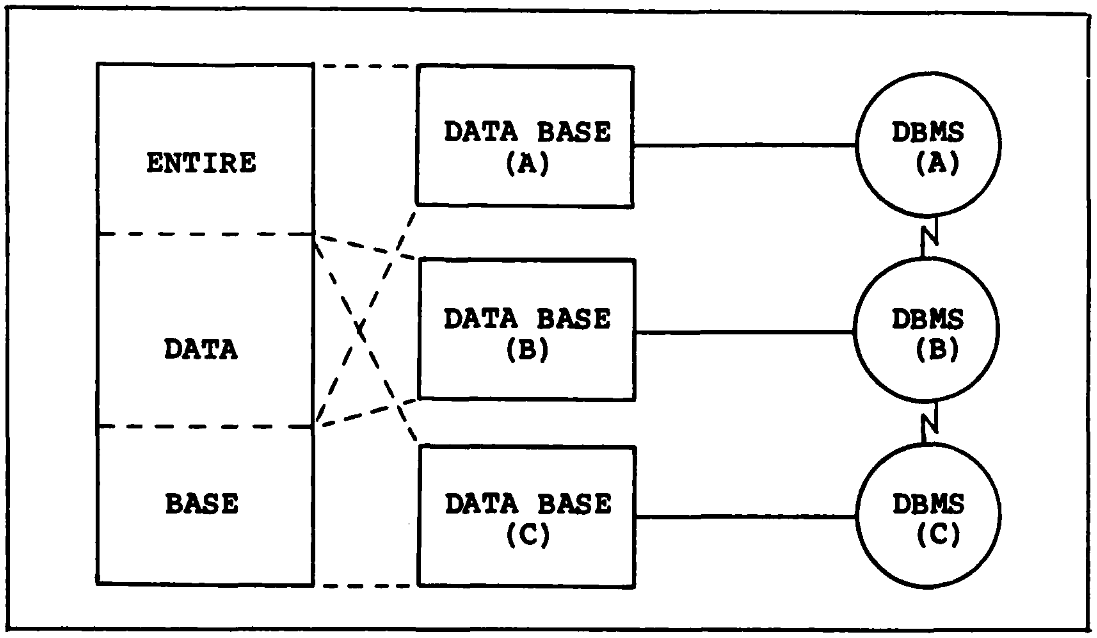


Figure 4 Replicated Data Base (Ref 5:50)

An extreme case could be that every computer has its own copy of the same data. In this scheme the strategy of what to duplicate and where is important. Ideally, it is possible to replicate just the right data at the right computers so that there are no misses. Three advantages are realized here. First, each site usually has access to data without contention. Second, there is a fast response time. Third, if a failure occurs, another copy of the data can generally be obtained. There are two important disadvantages in a replicated system. First, the duplication of data means a high storage requirement. Second, updates at each site containing the same data are expensive. Extensive communication among computers is required to maintain data consistency.

The problem of achieving optimum performance in distributed data base systems is complex. Table 1 summarizes a recommended data distribution method. The use of "high" and "low" for specificity are relative terms and must involve the communication costs and delays.

Mixing the partitioned and replicated approaches is an appealing alternative. For example, a data base might be mostly partitioned, with some computers having replicated data. A number of very complicated algorithms have been developed for determing where to store data to optimize performance. In one instance, an absolute optimizer and a heuristic optimizer were compared. They were given five sites and 20 data files to partition and replicate.

SPECIFICITY	REAL TIME UPDATE FREQUENCY	AMOUNT OF DATA	RECOMMENDED APPROACH
HIGH	LOW	SMALL	REPLICATE
LOW	LOW	SMALL	REPLICATE
HIGH	HIGH	SMALL	PARTITION
LOW	HIGH	SMALL	CENTRALIZE
HIGH	LOW	LARGE	PARTITION
LOW	LOW	LARGE	PARTITION
HIGH	HIGH	LARGE	PARTITION
LOW	HIGH	LARGE	CENTRALIZE

Table 1 Data Distribution Method (Ref 5:51)

Details such as access patterns and relocating costs for the data were provided. The result was that the heuristic program was much more effective in finding a solution (Ref 19:353). The cost of implementing data allocation algorithms must be considered when developing a DDBMS.

Directory Management

A directory in a distributed data base is a table of data locations throughout the system. Since data may be required from any other computer, the process of finding it is of fundamental importance. The question of where to locate a directory or multiple directories is discussed in this section.

There are three methods of directory systems used in computer networks. They are centralized, local, and distributed data directories. Choosing the method for any particular network depends on the computer network topology, communication cost, storage cost, directory query rate, and directory update rate. It is assumed that there is a local directory at each computer in the network. When one computer cannot locate the data it needs within its own local directory, it uses the directory management scheme of the computer network (Ref 6:442-457).

A centralized directory system has three approaches. They are called the single master directory, the extended centralized directory, and the multiple master directory system. In the single master directory approach, a master directory is located at only one computer in the network. If another computer requires data that is not listed in its local directory, it transmits a message to the master directory to find out where the data is located. There are three drawbacks to this approach. First, communication costs may become significant. Second, contention for directory access at the centralized location might cause significant delays. Third, the entire DDB is vulnerable to down time should the centralized location fail. Of course, the centralized directory must be updated when changes occur, but this is true of any approach. Updating in this approach is relatively easy.

In the extended centralized directory approach, the local computer again locates remote data by communicating with the central directory. However, now the local computer makes a copy of the remote data location in its own local directory. This information is now immediately available should the local computer ever need the remote data again. This reduces the communication cost but requires storage space in the local computer's directory. Additionally, the master directory records the fact that a local directory has copied a part of it. When the master directory requires updating, the information in other local directories can be updated also. The tradeoff in this more complicated process is based on expected remote data requests versus four new costs. First, the cost of storing the extended directory. Second, the cost of storing a "copied in a local directory" indicator in the centralized directory. Third, the cost of storing the more complicated update procedure. Fourth, the communication cost of updating local directories.

The multiple master directory is used in computer networks that have clustered groups of computers. In this instance, it is usually cost effective to have a master directory at each cluster. The obvious advantage is in reducing the communication cost for locating data. In addition, the extended centralized directory approach could be applied to any of the clusters.

The local directory system has no master or central directory available. When the local directory does not have the data location required by the local computer, a query is made to all the other computers in the system. This query may be in the form of a polling of each remote computer or a network broadcast, depending on the computer network configuration. In this system, copies of remote directory information are usually not made. Thus, the local directories are updated only be their owners which saves on update communication cost. However, this system has high communication cost for polling or broadcasting and is almost never preferred.

The distributed directory system has a master directory at each computer in the network. This provides a fast response time for locating data. The disadvantages are in the cost of storing the entire directory at each computer and the communication cost for updating each directory.

In evaluating the performance of these directory systems, there are four main considerations. They are communication cost, storage cost, query rate of the directory, and update rate of the directory. In an attempt to provide a guide in designing a directory system for a given distributed data base system, the following five conclusions from Chu (Ref 6:450-452) are presented.

(1) If the communication cost is much higher than the storge cost and with a low directory update rate (less than 10% of query rate), the distributed directory system is preferred over the centralized directory system.

(2) If the communication cost is much higher than the storage cost, with an update rate of greater than 15% of the query rate, the centralized directory system is preferred over the distributed directory system.

(3) The extended centralized directory system is preferred over the single master directory system at very low update rates (for example, less than 5 - 10% of the query rate).

(4) If network topology forms clusters and updating is greater than 5 - 10% of the query rate, multiple master directories at each cluster is preferred over the extended centralized directory system.

(5) When locating remote data, the centralized directory system is preferred over the local directory

system. However, if the update rate is greater than 50% of the query rate, and communication cost is lower than storage cost, the local directory system is the preferred method.

In general, even if storage cost is equal to or greater than communication cost, the results above are valid. Of course there will be exceptions depending upon the specific environment. Many different models of directory systems have been developed but only the more meaningful conclusions have been presented here.

The decision for which directory system to use depends on other characteristics of the network as well. Table 2 is a consolidation of the data base distribution methods (described in the previous section) and the directory location schemes described here. The centralized data base with a centralized directory system and a partitioned data base with a local directory system are usually the most desirable combinations because they tend to minimize storage cost, communication cost, or both.

Query Processing

The advantage of a distributed data base management system (DDBMS) over a centralized DBMS is the potential for access to a tremendous amount of data in a short period of time. In the distributed data base environment, the query processing strategy is the key to speedy data retrieval.

DATA BASE DISTRIBUTION METHOD	DIRECTORY LOCATION		
	CENTRALIZED	DISTRIBUTED	LOCAL
CENTRALIZED	FEASIBLE COMBINATION, MINIMUM STORAGE, MODERATE COMMUNICATION	FEASIBLE COMBINATION, MODERATE STORAGE, MODERATE COMMUNICATION	NOT A FEASIBLE COMBINATION
REPLICATED	FEASIBLE COMBINATION, MODERATE STORAGE, MODERATE COMMUNICATION	NOT A FEASIBLE COMBINATION	NOT A FEASIBLE COMBINATION
PARTITIONED	FEASIBLE COMBINATION, MODERATE STORAGE, MODERATE COMMUNICATION	FEASIBLE COMBINATION, MODERATE STORAGE, MODERATE COMMUNICATION	FEASIBLE COMBINATION, MINIMUM STORAGE, MINIMUM COMMUNICATION

TABLE 2 Characteristics of Data Base Distributions and Directory Location Combinations (Ref 16:147)

This strategy is usually an optimization process. The objective is to minimize the time required to process a query and provide the results to the user. To do this, the query is broken down to identify the data that is required. The data is then located using the directory system. Then a decision is made to determine how to process the data. This includes determining if specific data should be moved to another computer for processing. The query processing strategy also has the advantage of coordinating the processing of data at different computers in parallel. This strategy becomes the key to minimizing the query response time.

In discussing a query processing strategy, the following environment will be assumed. Each computer in the network will have a processing capability and its own data base. The data throughout the network will be considered in a relational format. The relational form is choosen for simplicity in discussing a query processing algorithm. The DDBMS, as well as redundant portions of the data base, are stored at each computer. A communication channel exists so that any computer can send data to another. An important assumption will be that the data transfer rate between a computer and its own storage element is significantly faster than the data transfer rate between computers on the communication channel. This assumption implies that minimizing the amount of data moved reduces the time required to move data. The data that is moved between computers will be in relational partitions. The physical distribution of relations is known in a system directory. The system directory will provide the location or multiple locations to find each relation required for a query's processing.

A query analysis is done when a user enters a query from any computer in the network. The query is usually in a high level query language. The query must be broken down to a set of interdependent subqueries. These subqueries are then arranged for execution in an ordered manner so that the result of each subquery produces either a final solution or another subquery which is used in future processing. The strategy here is to select a result location for each subquery so that subsequent processing occurs in the most cost beneficial order. The qoal is to ensure that the final transaction occurs at the originating computer.

In the relational format, a query is processed by using SELECT, PROJECT, and JOIN operations. The JOIN operation requires two relations to be combined to form a new relation. A JOIN required between two relations that are found in seperate computers will be called a COMBINING ARC. The solution to a query which requires combining arcs indicates that a relation needs to be moved from one computer to another (Ref 13:71).

The combination of local processing and data movements to solve the distributed query problem is called distribution (Ref 16:55). By using the SELECT and PROJECT operator, the local processing reduces the size of relations. The resulting smaller relations decrease data movement cost throughout the network.

Assume that the cost of data movement is much higher than the cost of processing data at a local computer. Under this condition, the first step of any distribution strategy is to reduce the size of all relations at their respective local computer by use of SELECT and PROJECT operators. This step should also perform all JOIN operations when both relations are located at the same computer. This is done before any data is moved.

To demonstrate how different distribution strategies work, a graphing method will be presented. It will utilize time as the unit of measure. The data movement time will be represented as a horizontal line (├──t──┤). The length of the line is an analog measure of data movement time. The vertical lines indicate the time required for local processing to reduce the size of a relation. A vertical line is used because local processing time is assumed to be insignificant when compared to data movement time. A query graph is made up of horizontal and vertical lines. Each relation in the query will be represented by a line of movement. Different lines of movement in a query graph demonstrate synchronization and parallel activities of query processing.

An Initial Feasible Solution (IFS) query graph is the simplest to consider (Figure 5). In this graph, a query is generated which uses relations located at other computers. Each relation, R_i, is reduced at the local computer (the first vertical line for each relation). Then the relations are moved, in time t_i, represented by the horizontal lines, to the computer where the query originated. The relations are then processed and the result provided directly to the user. In this example, R_3 is assumed to be at the computer where the result is required, therefore its line of movement, t_3, is of magnitude zero.

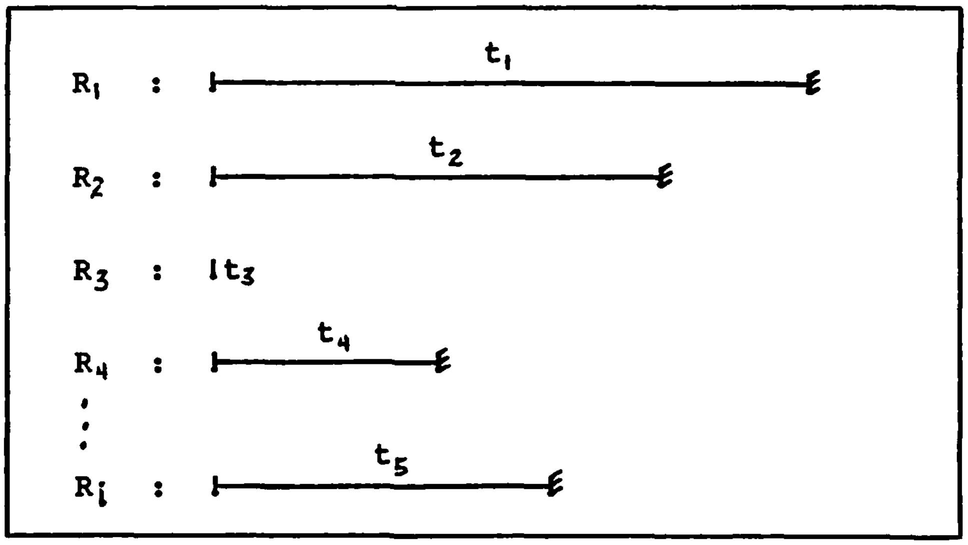


Figure 5 Initial Feasible Solution Query Graph (Ref 13:74)

Most distribution strategies use the IFS as a starting point for finding an efficient solution to a query. Improvement to the IFS is usually made by recursively finding data movements that reduce the size of relations, thereby reducing the time for subsequent data movements. Figure 6 demonstrates a solution where local processing is sequenced with serial and parallel data movements.

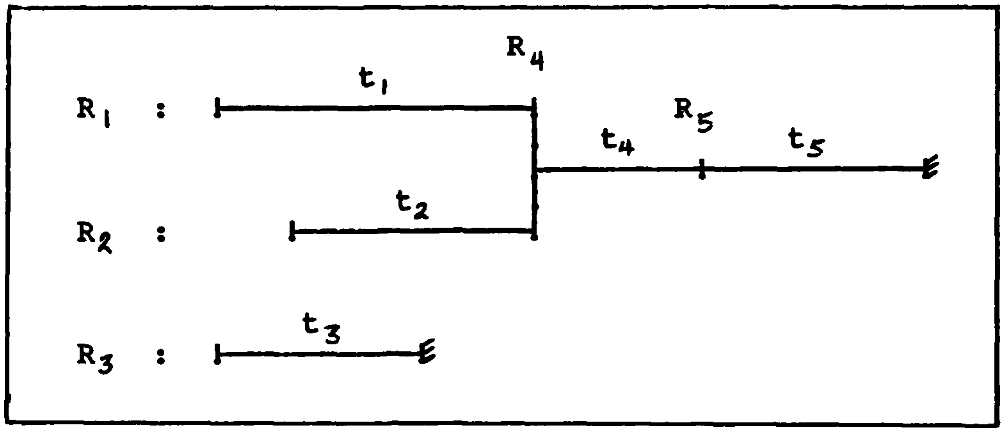


Figure 6 Parallel and Serial Data Movements on a Query Graph (Ref 13:74)

In this graph there are five relations and five line of movments. R_1 and R_2 are moved in parallel to R_4. These three relations are then joined and reduced as necessary. The result is then moved to R_5 where again the relations are joined and reduced. The result of that operation is then moved to the computer that originated the query. R_3 is seperately moved to the originating computer. R_3 and R_5 are joined at the originating computer and the result provided to the user.

Efficient distribution strategy is defined with two objectives of interest. The first is response time. In this objective, the time from the begining of query processing to the result being provided to the user is of interest. On a query graph, this is the time represented by a single horizontal line of movement across the width of the graph. The second objective of interest is total time. This is the total of all line of movements summed together. Figure 7 shows the difference graphically.

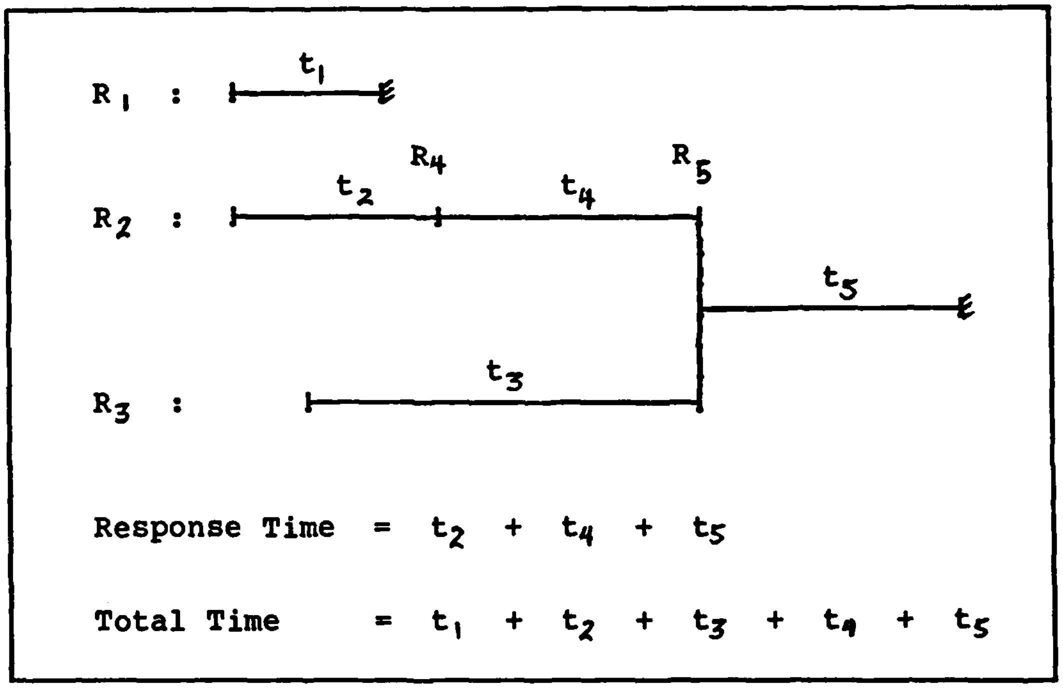

R_1 :
t_1
R_4
R_5
t_2
t_4
R_2 :
t_5
t_3
R_3 :
Response Time = $t_2 + t_4 + t_5$
Total Time = $t_1 + t_2 + t_3 + t_4 + t_5$

Figure 7 Comparison of Response Time and Total Time on a Query Graph (Ref 13:75)

The objective of interest depends upon the network. If the network is lightly loaded with few queueing delays, then minimizing the response time is preferred. If the network is heavily loaded then minimizing the total time may be the better objective. A very sophisticated DBMS might incorporate a switch to handle both loading conditions.

Research in the area of distributed query processing has developed many algorithms for data retrieval. An algorithm developed by Eugene Wong is now being implemented on SDD-1, a System for Distributed Data Bases (Ref 22:50-68). In this approach, the steps for processing the query alternate local processing and data movement. A recursive technique finds an optimum solution in the environment provided by SDD-1. Another algorithm by Hevner and Yoa implements query processing by moving small domains to large relations in order to reduce the size of relations that need to be moved (Ref 13:69-85). This technique is called Algorithm C. Under specific environments, one method will preform better than the other. Finding the optimum solution in a general query environment will require further research. Heuristic algorithms that use optimal movement strategies found in restricted environments might be the best approach. By first understanding the simplier strategies in a distributed data base system, it is hoped that solutions to more complicated situations can be realized.

Concurrency Control

Concurrency control in a distributed data base system addresses the problems of updating a multiple copy data base and synchronization of transactions. Synchronizing updates is vital in a distributed network in order to preserve consistency in the data bases. In centralized systems, concurrency control consists of locking data during updates. While the data is locked, attempts to read a record in that data are restricted until the lock is removed. In a distributed data base, the locking technique presents very difficult problems. The update problem in a DDB involves multiple copies of data stored at different computers. A change in a data item that has been stored at one computer must be changed at other computers. In this case, update messages need to be transmitted. Proper synchronization of this process is essential in avoiding incomplete or erroneous data. Simultaneous transactions such as those used in query processing also need to be controlled in order to provide correct results.

This section will present some terminology used when discussing concurrency control. The problem of deadlock is then discussed. Finally, the update control problem is discussed. Three approaches to solving the concurrency control problem in the DDB environment are presented. The first is a broadcast mechanism. The second is a time-stamp mechanism that is used in SDD-1 (a System for Distributed Databases). The third is called a majority voting algorithm. As the name implies, a request to update data in the network proceeds only when a majority of the individual data base management systems approve it. This algorithm uses a timestamping mechanism in synchronizing the voting and update action. The solutions to the update problem can then be applied to the problem of transaction synchronization.

Requirements and Terminology

Before presenting the different approaches to solving DDB synchronization problems, five requirements will be stated and the term "consistency" will be defined.

The following five concurrency control requirements must be met in order to provide successful operation in a true distributed environment (Ref 5:223).

(1) The cost of communicaton and processing in a concurrency control algorithm must be acceptable in terms of a given performance level.

(2) A locking protocol must have a deadlock prevention, avoidance, or detection and resolution mechanism.

(3) Precise timing or ordering of messages must not be manditory for successful operation. In the distributed environment, messages sent earlier than others may arrive later because of variations in communication transit time.

(4) If a computer or communication link fails, the rest of the network must be able to operate. Recovery to the correct state must be obtainable for failed computers becoming operational again.

(5) All computers must have update authority.

Under these requirements, the assumption is made that there are no "lost" messages (all known distributed concurrency control methods use this assumption.) To prevent lost messages, the sending computer saves a copy of the messages until acknowledgements are received. Messages which are not acknowledged are retransmitted until they are acknowledged or a failed computer is detected.

The meaning of the term consistency is divided into two levels. Strong consistency implies that all data is updated at the same time. This is desirable because each computer would then have the same update status at any time. However, implementation of strongly consistent DDBs means longer response time because the update of all copies are delayed until the last computer is available to proceed with the update. Weak consistency implies that the updates at different computers converge to the same update status over time. The tradeoff with weak consistency is a reduced delay time but with temporarily inconsistent data between computers. Different applications require different levels of consistency. Future discussions in this thesis are based on the concept of strong consistency (Ref 5:222).

A timestamp is a sequence number usually attached to update request messeges. There are two properties associated with the timestamp. First, no two timestamps have the same number. Second, each successively generated number is numerically higher. The timestamp is usually obtained by reading the value of a clock. Each computer in the network has the exact same time. To make the timestamp unique at the global level in the network, only one number can be generated at each computer per clock change. Each timestamp is a concatenation of a predefined computer address and the clock time. Thus, even if two timestamps have the exact same clock time, they can still be used to serialize updates within the network (Ref 5:231).

The Deadlock Issue

Since many of the concurrency control algorithms use a lock mechanism, it is best to have an understanding of deadlock. (In this discussion, a resource may be a data file and a process may be an update request in a distributed data base environment.) Deadlock is a situation where two or more processes require resources held by the other(s). Each process has locked its own resource and cannot proceed without the resource of another process. Figure 8 demonstrates the deadlock issue. In a distributed data base system this problem is much more difficult to handle than on a centralized system. For deadlock to occur, the following four conditions are necessary:

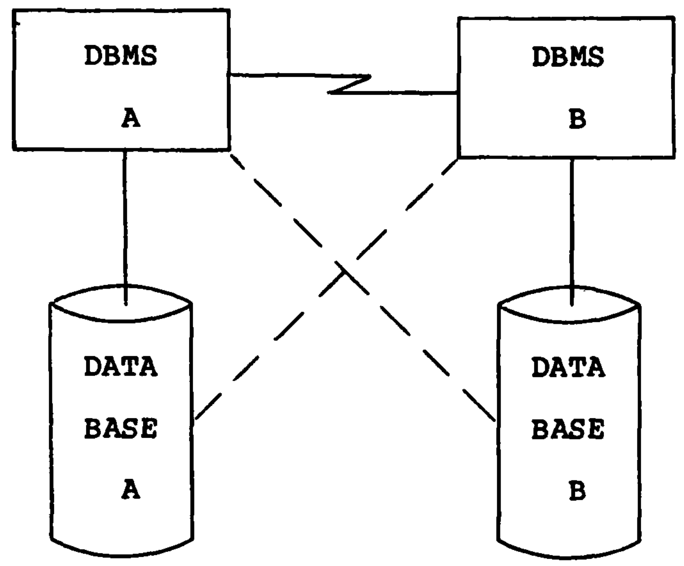
(1) Concurrent processes;

(2) Mutually exclusive access;

(3) Nonpreemptive scheduling;

(4) Partial resource allocation (Ref 5:245).

When deadlock occurs, at least two processes must be concurrently trying to claim resources. Mutually exclusive



(1) DBMS A has locked data base A in preparation for an update.

(2) DBMS B has locked data base B in preparation for an update.

(3) DBMS A requires data base B to complete its update.

(4) DBMS B requires data base A to complete its update.

(5) Since both data bases have been locked, neither can complete its update.

Figure 8 Deadlock Condition (Ref 5:243)

access implies denying access to all but one process. Nonpreemptive scheduling means preventing a resource from being unlocked unilaterally. Partial resource allocation implies that a process may wait indefinitely for a resource. Deadlock occurs only if all four conditions exist and a stalemate occurs as described earlier. If any one of the four conditions is excluded, deadlock can be avoided. But if any condition is excluded, a DDB system would be seriously affected. The capability of concurrent processing in a DDB system is essential to provide efficient operation. To avoid inconsistent data created by concurrent updates, mutually exclusive access is required. Nonpreemptive scheduling is necessary because updates are often required at several computers. To provide strong consistency, update synchronization must not be terminated prematurely at remote computers. Partial resource allocation provides more efficient use of resources because they are assigned where needed (Ref 5:245-246). The strategies for handling the deadlock issue are prevention, avoidance, and detection with resolution.

Prevention is very difficult because a process must request all required resources before it can start. Knowing all required resources is usually not possible. Additionally, holding these resources would be inefficient because of idle time as well as decreasing system concurrency. If prevention is used in a DDB environment, it is realized by eliminating one of the four conditions necessary for deadlock.

Avoidance requires knowledge of resources needed by a process to run to completion. In a DDB, this approach is unattractive because the advance information is usually not available or is distributed throughout the network. In this case, the overhead to avoid deadlocks would cause unacceptable delays.

Detection is made by searching for cycles in a Wait-For Graph. Figure 9 shows an example of a wait-for graph. The nodes are labeled by the pair (transaction , site). A directed arc indicates that a given transaction requires a resource at the node being pointed to. Figure 9 shows a network of two sites, S1 and S2. It shows two deadlock cycles. One of them is internal to site 2. It follows the path (T3 , S2), (T5 , S2), (T6 , S2). The other cycle spans both sites and is known as a global deadlock. It follows the path (T3 , S1), (T3 , S2), (T4 , S2), (T4 , S1). In a local system, building and maintaining the wait-for graphs is usually effective. In a distributed data base, global wait-for graphs are rarely efficient. Two alternatives are hierarchical and distributed detection (Ref 18:95-112).

In the hierarchical approach, the network is partitioned into subnetworks, each with its own deadlock detection controller. Additional deadlock detection controllers are placed at higher levels within the network hierarchy. Controllers that are at the bottom level of

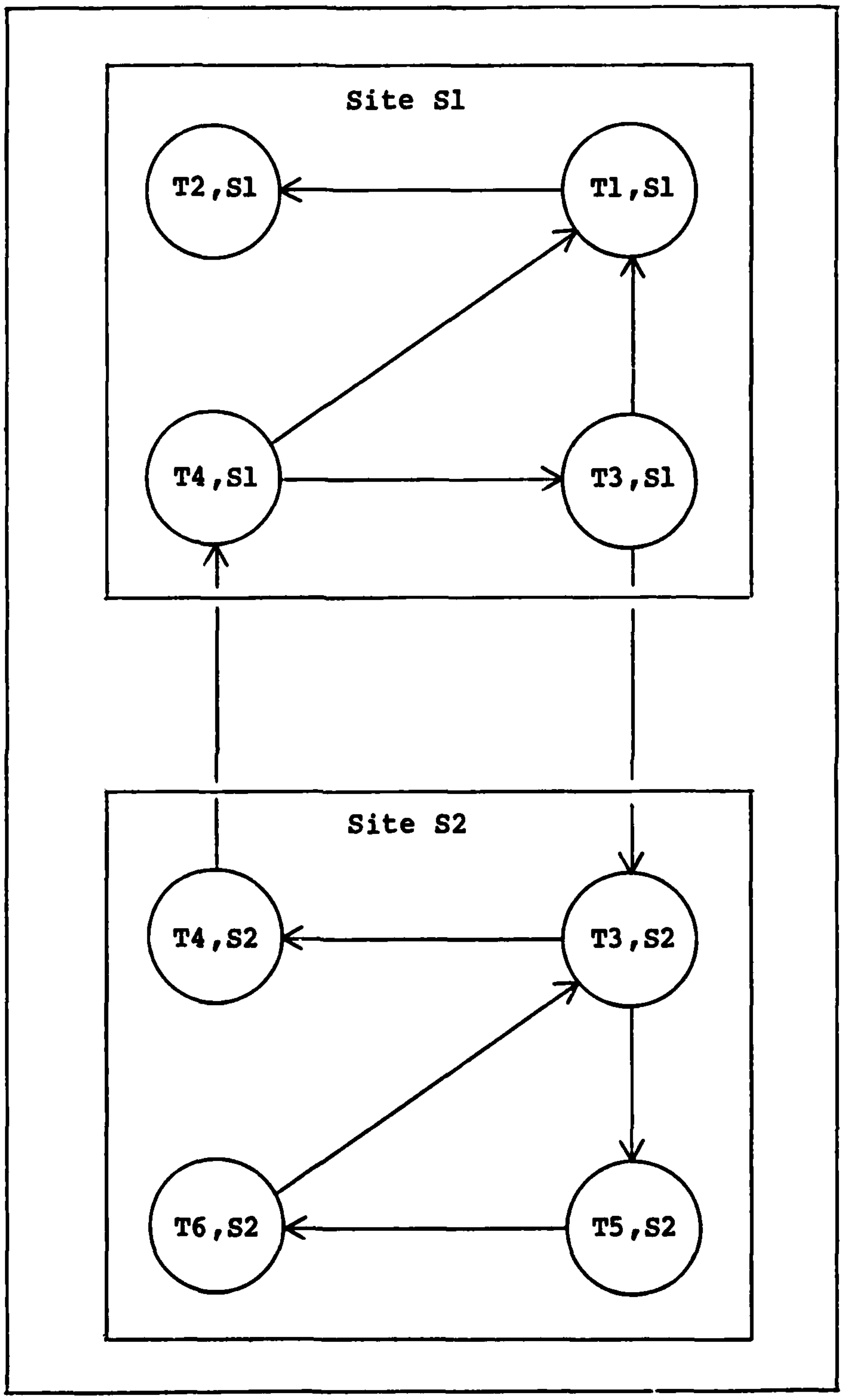


Figure 9 A Wait-For graph for a Network with Two Sites (Ref 18:98)

the hierarchy are called leaf controllers (LKs). Those higher up are called nonleaf controllers (NLKs). Figure 10 shows a network with three leaf controllers (LK1 , LK2 , LK3), and two nonleaf controllers (NLK1 , NLK2). Each controller keeps a wait-for graph for its section of the network. This reduces the detection responsibility and also contains a deadlock occurance to a specific area.

In the distributed approach, each computer maintains a local deadlock controller. Transactions between computers are made by sending requests directly to the remote deadlock controller. The remote controller compares requested resources and either accepts or rejects the request. An advantage of distributed data bases is apparent here because if the data is replicated at another computer, the request can be transferred.

Broadcast Concurrency Control

The broadcast concurrency control mechanism updates all other computer data bases simultaneously. When an update is required, the local computer checks its "local lock table." If the data is not locked, the local computer transmits a request to all other computers to lock the required data. If any computer detects a conflict, the update is suspended. If all computers acknowledge positively, the local computer transmits the update. When all computers have completed the action, another acknowledgement is transmitted from each remote computer.

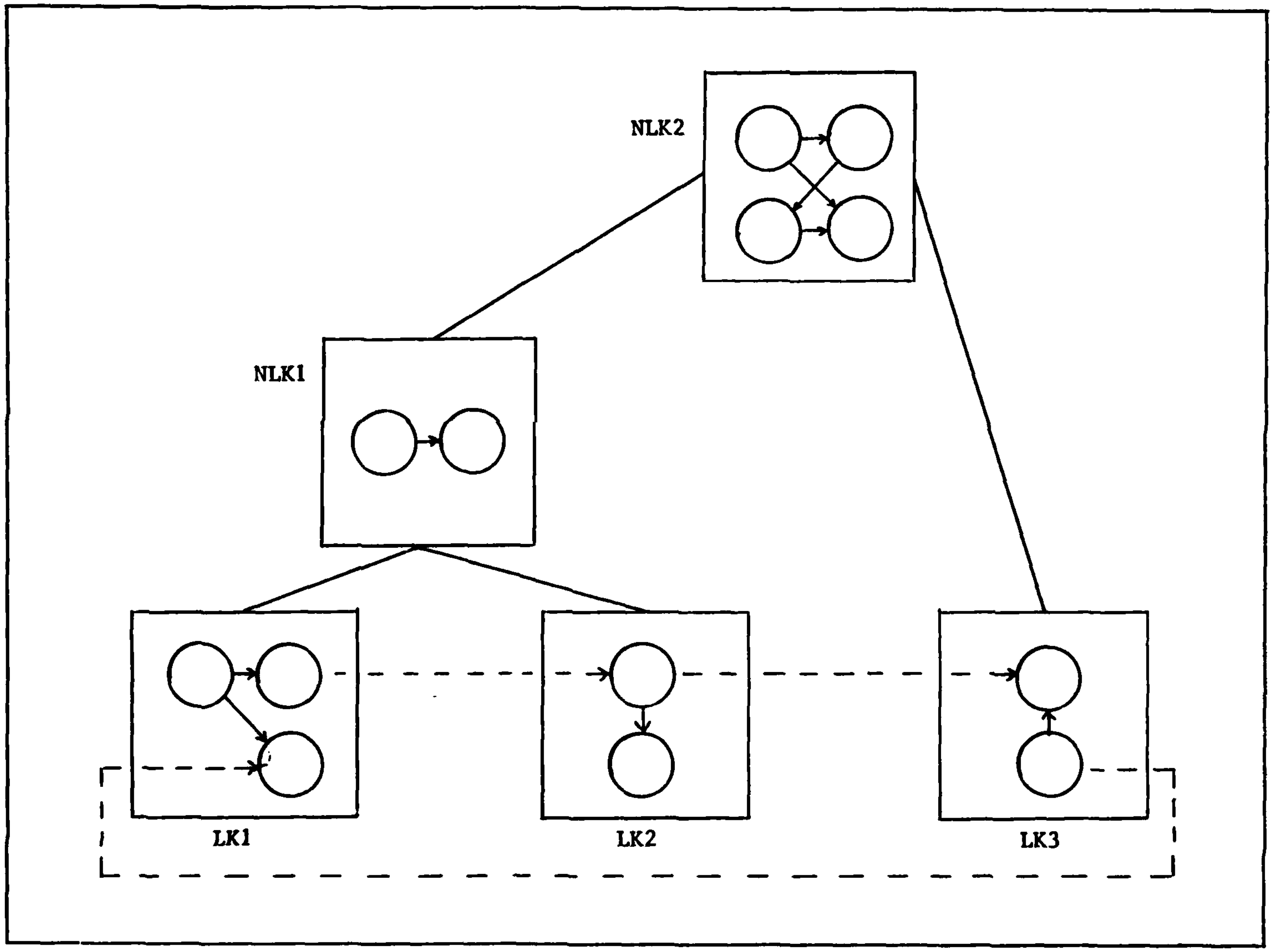


Figure 10 A Hierarchical Deadlock Detection Example (Ref 18:100)

Each computer maintains a local lock table. Each lock request, whether locally or remotely generated, is checked with the local lock table.

A time out counter is used to detect failures. A computer is assumed to have failed if a response is not received by the time out period. If this occurs, the failed computer address is recorded at all other computers in the network. Only operational computers continue to communicate with each other. Each computer keeps a list of unavailable computers. As long as the list is not empty, each computer keeps a history of the updates it performed. When a failed computer rejoins the network, it contacts all the other computers and receives the necessary updates.

Timestamp Concurrency Control

The SDD-1 (a System for Distributed Databases) system is used to present the timestamp approach to concurrency control. SDD-1 uses a priori knowledge of transaction types to form four efficient synchronization processes called protocols. This approach sometimes avoids locking of the data at all computers which need to be updated. It is also designed to be deadlock free. In designing the system, the decision was made to run different synchronization methods with specific transaction types. The design was then made into a table which can be accessed during execution to determine the best synchronization method without intercomputer communication. The transaction types are characterized by "read-set" and "write-set" data items. Read-set transactions read data from a group of predefined sites. Write-set transactions write data to predefined sites. There may be many combinations of sites that form different groups. Thus having 100 transaction types would not be unreasonable in a small network.

The four synchronization protocols are arranged by degree of complexity. Protocol one handles transactions with no delay in processing. Protocol four handles completely general transactions with a processing procedure as costly as global data base locking.

Protocol one locks only local atomic data before initiating an update. After the update is made locally, the update is broadcast to other computers. Each computer that receives the broadcast acknowledges the update. Once acknowledgments are received, the local data is unlocked. This protocol works because the data that is locked locally is known, ahead of time, to have no effect on actions at other sites.

Protocols two and three are specialized. Protocol two is used strictly for retrieval transactions. Protocal three is used for update transactions. In both of these protocols, a waiting state occurs for processes to take place at predetermined time intervals.

Protocol four handles general transactions that were not anticipated ahead of time. It requires extensive synchronization and is expensive. It operates according to the following five steps.

(1) A local computer needs to update data in the network. It picks a time in the future, "T", when it estimates that all the other computers will be able to complete the synchronization step of locking the data.

(2) The local computer transmits the lock request with a special timestamp field set to "T" and the data variables required for the update.

(3) When the remote computers receive the lock request, they check their own clock with the time "T". If a remote computer's clock is later than "T" when it receives the message, it transmits a reject lock message to the originator. If the clock is before "T", then it acknowledges the message and records time "T". It then prevents any local transaction from accessing the intended update data variables until time "T".

(4) When all computers acknowledge the request for an update, the local computer transmits the update to all the other computers and updates its own data base at time "T".

(5) The remote computers update their data bases at time "T" and release their locks.

Choosing the most efficient protocol for each type of transaction is based on a mathematical analysis of different transactions interfacing with each other. This particular discussion on SDD-1 timestamping methodogy takes advantage of specific knowledge of the transaction environment. The protocol selections for the same transactions in a different environment would be different.

Majority Consenses Concurrency Control

The majority consensus algorithm is another solution for distributed data base update synchronization (Ref 21:88-94). In this method, the individual database management systems (DBMS) vote to accept or reject an update request. If a majority vote to accept the request, then it is applied to all copies of the data in question. Information used in voting is checked for validity by using timestamps. Applying an update is also verified by checking timestamps.

The majority consensus algorithm shares the responsibility of processing updates with all the DBMSs that are involved. When an update is required, the DBMS that receives the change builds a request that is forwarded to the other DBMSs. The request is processed by using rules established to govern the voting and updating. The following six steps are used in the update procedure:

(1) An update request is decomposed at the DBMS to obtain the data elements required for the update. The timestamps stored with these elements are also extracted.

(2) An update request is created by computing new values for the data elements and attaching the extracted timestamp.

(3) The request is forwarded to all other DBMSs which have a copy of the data in question.

(4) The entire network cooperates to synchronize the update. If a DBMS is involved with the decision at hand, then it participates in the voting procedure.

(5) If the request is approved by a majority of the DBMSs which are involved then the update is made.

(6) If the request is rejected, then the originating DBMS may resubmit the request (Ref 21:89).

The properties of majority subsets used in conjunction with timestamps prevents conflicting update requests from being accepted. The algorithm works with the following four rules:

(1) DBMS to DBMS communication rule;

(2) Voting rule;

(3) Request resolution rule;

(4) Update application rule (Ref 21:89-92).

The Communication Rule

The communication rule described here assumes a daisy chain communication discipline. Figure 11 shows two possible adaptations of the daisy chain principle. In the "pure" method, a DBMS that receives an update request votes and forwards the request to the next DBMS in line. This process continues until the request is resolved. In the "timeout" method, the ability exists to bypass a DBMS when necessary. An update request may be abnormally halted

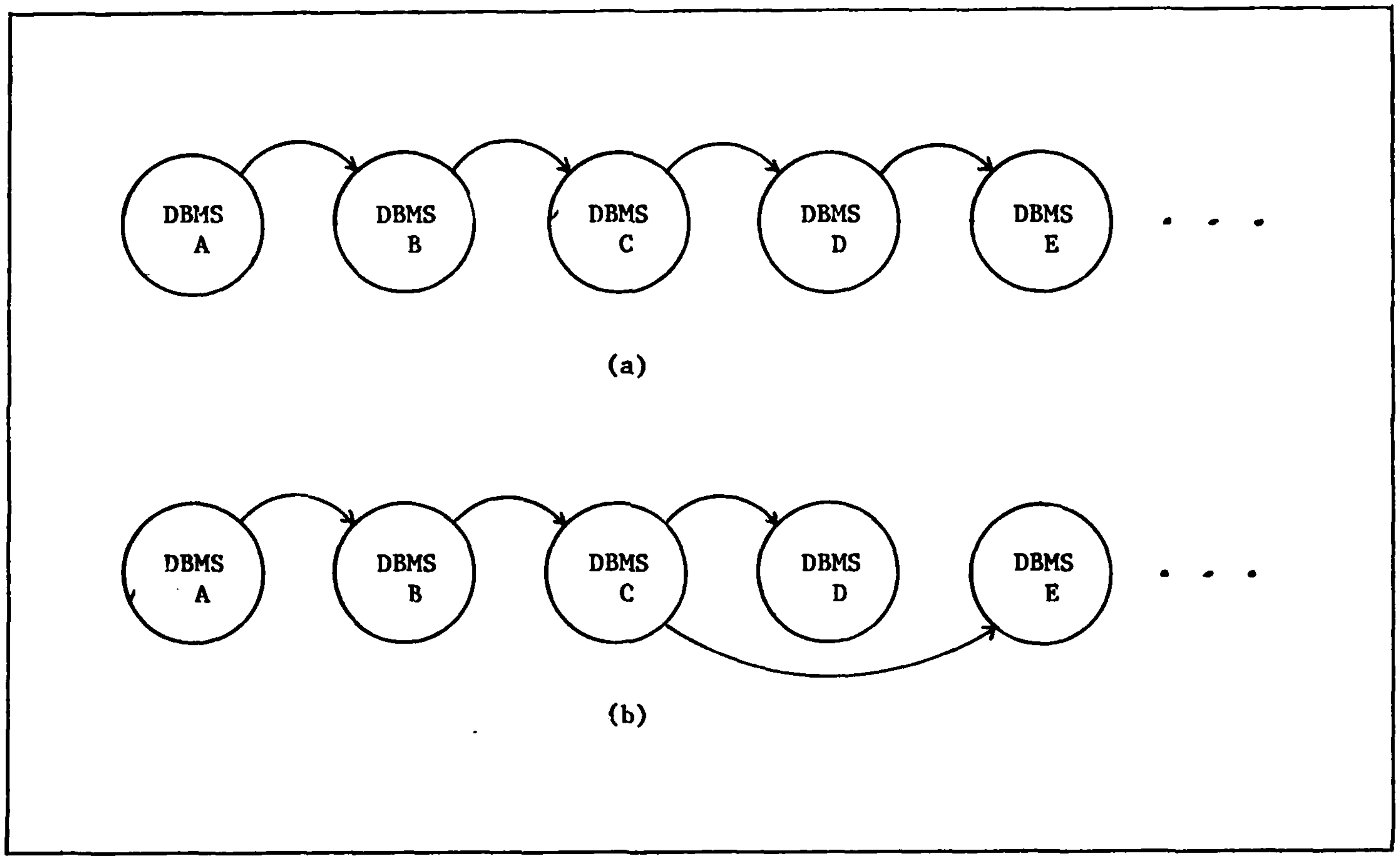


Figure 11 (a) Pure Daisy Chain
(b) Daisy Chain with Timeout and Retransmit (Ref 21:90)

using either of these two procedures. Consider a DBMS which has an unresolved request and cannot locate another DBMS which has not voted because of communication or computer failures. In this case nothing can be done until a majority of the DBMSs have voted to resolve the request. Another condition may arise which the timeout method can solve but the pure method cannot solve. Suppose a DBMS which currently has an unresolved request under consideration fails and cannot forward the request. In the pure method the update processing halts. The timeout method provides a backup procedure. This procedure involves DBMS-C, in Figure 11b, setting a timer to check with DBMS-D to which it forwards a request. If DBMS-C receives an acknowledgement from DBMS-D either before the timeout or during the check then all is well. If DBMS-C cannot communicate with DBMS-D, then it acts to forward the request to DBMS-E. This procedure contributes to the robustness of the algorithm by ensuring that the update request progresses further toward resolution.

The Voting Rule

The same voting rule is used by each DBMS. The idea is to allow nonconflicting updates to proceed while resolving updates that would create inconsistent data. To do this, a DBMS checks to determine if the intended update will change data that has been modified since the request was constructed. If the data in the request has not been modified, then the premise upon which the update is based is valid. In this case the DBMS votes "OK" to accept the request. If the data has been modified, then the premise to change the data is invalid. In this case the DBMS votes "REJ" to reject the request. By checking the timestamps, the currency of the data can be determined.

Another type of conflict can occur which complicates the voting rule. It occurs when two DBMSs concurrently initiate conflicting requests. The possiblity exists for one DBMS to consider a request that has valid current data but conflicts with a pending request. In this case the DBMS votes "PASS" on the second request.

In summary, the voting rule proceeds as follows:

(1) Compare timestamps from the update request and the local database.

(2a) If local data is obsolete, vote "REJ".

(2b) If local data is current and does not conflict with a pending request, vote "OK".

(2c) If local data is current and does conflict with a pending request, vote "PASS".

A "PASS" vote is reconsidered later or is resolved by a majority of the other DBMSs voting "REJ" or "OK" later.

The Request Resolution Rule

To determine if a particuliar DBMS's vote has resolved a request, it uses the request resolution rule immediately after determining its vote. For the daisy chain discipline the rule has two parts. Part one involves what goes on after voting and has the following four possibilities:

(1) The vote was "OK" and a majority consensus now exists. For this condition, accept the request and notify all the other DBMSs that the request was accepted.

(2) The vote was "REJ" and a majority consensus for accepting the request no longer exists. For this condition, reject the request and notify all the other DBMSs that the request was rejected.

(3) The vote was "PASS" and a majority consensus for accepting the request no longer exists. Again, reject the request and notify all the other DBMSs that the request was rejected.

(4) If "REJ" or "PASS" was voted and the possiblity still exists for a majority consensus to accept the request, forward the request and the vote tally to a DBMS that has not voted on the request yet.

Part two involves what to do after learning that a request has been resolved. Its two possiblities are:

(1) The request was accepted, thus apply the update using the update application rule and reject conflicting requests that were deferred because of the request.

(2) The request was rejected, thus reconsider conflicting requests that were deferred because of the request.

The Update Application Rule

The update application rule provides a basis for accepting "properly sequenced" updates. Because of majority consensus, it is possible to receive notification of the acceptance of an update which has since become obsolete. This is possible because of the "PASS" vote option. A DBMS that has voted "PASS" on two requests might eventually receive notification to accept both requests. But because of communication delays, the notification might be received in an undesirable order. Therefore the timestamp is used again to prevent old data from replacing new data.

Majority Consensus Algorithm Conclusions

The following five properties of the majority consensus algorithm are observed:

(1) The distributed data base network either accepts or rejects an update request.

(2) The algorithm prevents deadlock.

(3) The data converges to the correct state. The speed of this convergence depends on the number of copies of the data and at which computers it is stored.

(4) Internal data consistency is preserved. Inotherwords, the algorithm has the "effect" of locking data while updates are resolved.

(5) Sequencing anomalies are avoided because of timestamping procedures.

The algorithm also functions effectively in case of minor communication failures because the responsiblilty for processing updates is distributed throughout the network.

The communication and computational costs are not significantly greater than other concurrence control approaches. There is, however, a small short term storage requirement for remembering the state of pending update requests (Ref 21:92).

Reliability

The issue of reliability in a distributed data base system is concerned with ensuring that reliable operation continues when a computer fails or a communication failure occurs. As long as a network exists which can produce meaningful results, that part of the network should continue to operate. The "certain" loss of individual sites during operation should not cause the entire network to halt. Reliability also addresses the issue of subsequent recovery of sites which have failed and are again ready to join the network (Ref 11:645-647).

The replicated distributed data base system will be used as the model for further discussion. The types of failures are computer and communication failures. Two options are available when the computer or communication link becomes operational again. These options depend on the length of time of the outage. The outage time can be defined as a predetermined time period or the length of an audit trail. An audit trail is a file containing all changes to a data base which has enough information that a recovering computer can incorperate those changes into its own version of the data base. For short outages, the usual recovery procedure is to use the audit trails of other computers in coordination with the data directory system to update its data base. For long outages, the most efficient recovery procedure is usually accomplished by rebuilding the data base by acquiring current replicated data from other computers. A search of an audit trail might be required if some portions of the data base cannot be replicated to the current system time.

Most reliability algorithms assume that a failure is detected when it occurs or when the first access to a failed computer is made. Additionally, it is usually assumed that a computer does not generate erroneous messages. In either case, no techniques are available for coping with failures that do not satisfy these assumptions because of complexity.

Communication failures often involve missing messages, out of order messages, physical line failures, and garbled messages. These failures can be detected by using techniques such as sequence numbers, rerouting messages, and redundancy checks or acknowledgements (Ref 5:275-285).

In some distributed data base systems the computers monitor each other by exchanging status messages. This may be done at constant time intervals or whenever there is a question as to whether a computer is down. The message usually includes the computer address, a sequence number, and a timestamp. Any computer which fails in this environment is known to the rest of the network and recovery procedures as well as avoidance procedures are instituted (Ref 11:647).

Integrity

Integrity in a distributed data base system refers to the accuracy, validity, consistency, and availability of the data. Higher (better) degrees of integrity require higher costs to implement.

The accuracy of any data base depends on the correctness of the data as it is entered into the data base. This type of integrity is usually accounted for at the local computer level. Application programs must be written which minimize errors input by the users.

Validity refers to the data values and associated formats. This ensures that data values conform to the data base definition, not that it is accurate. For example, illegal ranges of data values must be flagged as errors and corrected. Also, data that requires numeric fields should not have alphabetic characters in them (Ref 3:8).

Consistency refers to the effectiveness of concurrency control. For example, a data value in two different places must be the same during operation.

Availability refers to the probability that data in a computer is available at a specific time. This is closely related to the concept of reliability. A mathematical definition of availability is described as follows:

$$A = \frac{T_F}{T_F + T_R} \qquad \text{and}$$

$$U = 1 - A \qquad \text{where,}$$

A	:	Availability;
T_F	:	Mean Time to Failure;
T_R	:	Mean Time to Repair (Recovery);
U	:	Unavailability.

The availability of any computer in a network is defined as the probability of its successful operation within that network (Ref 16:166).

Security

When data is shared with other computers in a distributed data base network, the problem arises of keeping specific information confidential. A secondary but equally important problem also occurs in the distributed environment. Once confidential information is accessed, the receiver must be informed that the data is to remain restricted. The solutions to security problems in this environment are usually extensions of existing methods used in centralized systems. These methods include access decisions, access authorizations and data encryption.

Access decisions are those providing controlled access to the data base. The use of passwords is a typical device. Password effectiveness depends on the user to maintain secrecy. The ability must be provided to change a password whenever a user feels it has been compromised. After entering a password a user then has access to predetermined areas of the data base. A sequence of passwords may provide further access to different levels of the distributed data base.

Access authorization is used to check if a user is allowed access to specific parts of the data base. This is usually done by keeping an authorization file or table consisting of the user identification, object data, and access privileges. The access privileges are sometimes categorized as combinations of no access, read, write, and modify. As a general rule, access control and security checks should be made as soon as possible to prevent unnecessary computations. When a query spans multiple sites, each computer should verify the access before further processing.

Encryption in a distributed network is sometimes necessary to prevent unauthorized access. By alteration of the data, it becomes unrecognizable to everyone except the users. Only the users can reverse the alteration process. Three usual encryption schemes are item value encoding, transposition and substitution. Item value encoding involves replacing human readable words with codes. An example might be "Male = 1" and "Female = 2".

Transposition is a reording of characters according to a predetermined rule. Only the sender and receiver know the rule for reversing the process. Substitution is done by replacing characters of text with a keyed ciphered alphabet. Encryption is usually only feasible along the communication channels and only with sensitive data (Ref 5:257-273).

Summary

There are many types of DDB systems. Each has been developed for the environment which it will be used. This chapter pointed out that the architecture strategy is significant in determining the complexity of the system. There are three basic approaches to DDB architecture. They are, from simpler to more complex, integrated, homogenous, and heterogenous.

The three data allocation methods generally used are centralized, partitioned, and replicated. Query processing optimization can be most efficient in a replicated scheme. However, the recommended approach depends on specificity, update frequency, and the amount of data.

The three directory management approaches are centralized, local, and distributed. Choosing the method for any particular network depends on the computer network topology, communication cost, storage cost, directory query rate, and directory update rate.

Query processing is the key to speedy data retrieval.

Optimization of query processing requires parallel data activities such as data reductions and data movement. Improvements to an Initial Feasible Solution are made by recursively finding data movements that reduce the size of relations, thereby reducing the time for subsequent data movements.

The concurrency control issue addresses the problems of updating a multiple copy data base and synchronization of transactions. Three solutions to these issues are a broadcast mechanism, a timestamp mechanism, and a majority voting algorithm. One method can not be called better than the others because the network environment provides different advantages and disadvantages for each.

Reliability in a DDB system concerns computer failures and subsequent recovery procedures. In a replicated data environment, the problem of data updates missed by a failed computer become important. In some cases an audit trail is used to update the data during recovery. In other cases the data is copied from current replicated data. Detection of failed computers is essential, therefore reliability algorithms usually include a network monitoring activity.

Integrity in a DDB system refers to the accuracy, validity, consistency, and availability of the data. Each of these factors contribute to the degree of integrity in a DDB system. Generally, higher (better) degrees of integrity require higher costs to implement.

Security problems in a DDB system are usually extensions of existing methods used in centralized systems. These methods include access decisions, access authorizations and data encryption.

III. REQUIREMENTS

Introduction

This chapter presents long term and short term requirements for a distributed data base (DDB) system in the digital engineering laboratory (DEL) at AFIT. Long term represents three or more years from now. The long term requirements will be established first. The short term requirements will then be set so as to become a building block for the long term plan.

The purpose of the DEL is to provide an educational environment for students to gain experience and do research on different computer systems. Thus the long term objective of this specific project is to provide a pedagogical tool for future classes studying DDB systems. The first few years of this project will emphasize current technology in the design and development of a DDB. Once the development reaches a stage where data is successfully being exchanged, the effort will be transformed into a pedagogical tool. Hands on experience on a DDB system could be provided to students in one of the data base courses taught here at AFIT.

As an operational DDB in the DEL is realized, individuals at AFIT may see solutions to some of their requirements. Eventually, a DDB could be established which extends beyond the DEL and links with AFIT computers in other schools. This could also provide a tool for AFIT management and administration.

Hardware Considerations

The DEL currently includes many individual computer systems. Among them are the VAX 11/780, Heath H89, an elementary network of six DEC LSI-11/02's, two PDP-11/10s, a Data General Eclipse, and other small computers. The first three are specifically supporting, though not strictly, data base research in some form.

The VAX 11/780 implements TOTAL, a commercially available network data base management system (DBMS). The VAX also implements RIM, a Government-owned relational DBMS. Future intended enhancements include adding an EMBRESS DBMS under the UNIX operating system as well as a US Army developed heirarchical DBMS.

The Heath H89 computer operates a relational DBMS known as dBase II.

A network of eight LSI-11 and PDP-11 computers, known as LSINET, is currently supporting two relational data base efforts. The first is the development of an optimal relational data base system for use on a single microcomputer (Ref 20). The second is in the area of specialized backend data base architectures in which hardware components are being simulated by the network of LSI-11 microcomputers (Ref 9).

In addition to the computers listed above, there are thesis efforts currently ongoing which will provide a communication link between all the computers in the DEL. This will be known as DELNET (Refs 11,14). This particular communication link will also include a Harris 500 computer, implementing TOTAL, at AFIT/LS.

It must be mentioned that nearly all computer systems are always undergoing some kind of change. This is especially true for the computer systems in the DEL. Changes such as software upgrading, replacement hardware or additional hardware being bought are factors which should be considered.

Long Term Requirements

The long term goal of a DDB system in the DEL is to provide a communication link between different types of data bases on different computers so that they can exchange information. This describes a heterogenous DDB. As an example, the VAX might operate with a network DBMS, the Heath H89 might operate with a relational DBMS and the LSINET might operate with a different relational DBMS. This system should be allowed to switch the data and DBMS on any one computer and still operate with no effect on the other computers. For instance, in the example above, the VAX might switch its network data base and DBMS to a heir-archical DBMS with a different data base. This change should only effect the specific data available from the VAX to other computers in the DDB. No software or hardware changes should be required at any other computer to allow this change.

The specific issue of a communication language needs to be resolved. A long term research goal could be to develop a common or universal language for intercomputer communication. This would require a translation from any DBMS to the universal language and vice versa for each computer in the DDB network. The alternative is a language translator at each computer which can translate to any of the other DBMSs in the network. Since the long range plan includes numerous types of data bases and DBMSs, the combinations of translations for this alternative may be impractical.

The "data base" will be considered as the sum accumulation of all the individual data bases in the network. The DDB network should not be restricted to only computers in the DEL, since there are data bases on other computers at AFIT. Also, access to computers and networks outside of AFIT such as ARPANET should be considered. The number of computers attached to the network should be restricted only by the memory available to store all the other computer addresses and data elements contained within each. In other words, selecting the distributed query processing procedure through the use of a directory must be maintainable. The physical distribution of the data should be invisible to the user. However, all data must be conceptually available to all users.

Redundant data must be allowed so that the advantages of a DDB can be realized. This includes operation where the same data is available from two different types of DBMSs.

For example, a "Parts" inventory may be stored in a heirarchical data base on one computer. The same "Parts" inventory may also be stored in a network data base on another computer. This implies some form of concurrency control to maintain integrety.

A deadlock avoidance system is mandatory. In a DDB system this is particuliarly difficult because several computers must be coordinated together without slowing their progress.

The pedagogical long term requirements necessitate that a protection mechanism be incorporated into data bases identified for operational uses. Student access should otherwise be as flexible as possible. Student projects to add a small data base and DBMS into the DDB system should be a realistic exercise within the time frame of a single scheduled class (ten weeks).

Short Term Requirements

The short term requirements in this section are intended as guidelines for the next one or two years of research and development. The network of eight LSI-11 and PDP-11 computers will be used to create a distributed data base management system. This will provide an easily accessible test bed for development of the software required for a DDB system. The network will be developed as a simulation of a heterogenous data base system. Each microcomputer will have a data base and DBMS simulated on it. As an example, one microcomputer might simulate a network DBMS, another might simulate a heirarchical DBMS and another might simulate a relational DBMS. The simulations should be simple so that emphasis can be placed on the distributed data base management system.

Each microcomputer in the LSINET will need to have its own distributed data base management (DDBMS) operating module. This module should link the data base stored at that microcomputer with the other data bases in the LSINET. The module needs to contain all the DDBMS functions necessary for operation. It should include the following subprograms:

(1) a linkage program between the network and the specific DBMS to which it is connected;

(2) a directory program for computer addresses and data elements within the network;

(3) a distributed query processing procedure;

(4) a "redundant data" processor;

(5) a concurrency control processor;

(6) a deadlock avoidance processor;

(7) a reliablity processor for adding or dropping computers from the network.

These short term requirements are similiar to the long term requirements except that the network is restricted to the LSINET. However, once the logical scheme of the DDBMS operating module is working properly on the LSINET, then coding changes required to transport this management system to the DELNET should be limited to the linkage program.

IV. SYSTEM DESIGN

Introduction

The design of this distributed data base system management system (DDBMS) is described in this chapter. For pedagogical and research purposes, strong emphasis will be placed on functional modularity so that some components can be changed without affecting others. The components that would be necessary in a heterogenous architecture environment will be identified. A heterogenous environment has been deliberately selected so that the most general purpose DDBMS can be developed. A general purpose DDBMS has not been previously developed although many computer laboratories are actively working on the subject. The LSINET in the DEL has been targeted for initial coding and testing of this project. The reason for choosing the LSINET is because it provides a readily accessible testbed for what will undoubtedly be a complex software development effort.

Components of the DDBMS

To describe the components of this distributed data base system a single computer will be examined. This computer may implement on the local level a relational, hierarchical, or network data base. Figure 12 illustrates the different components which this typical computer will have. Figure 12 contains three components with the word

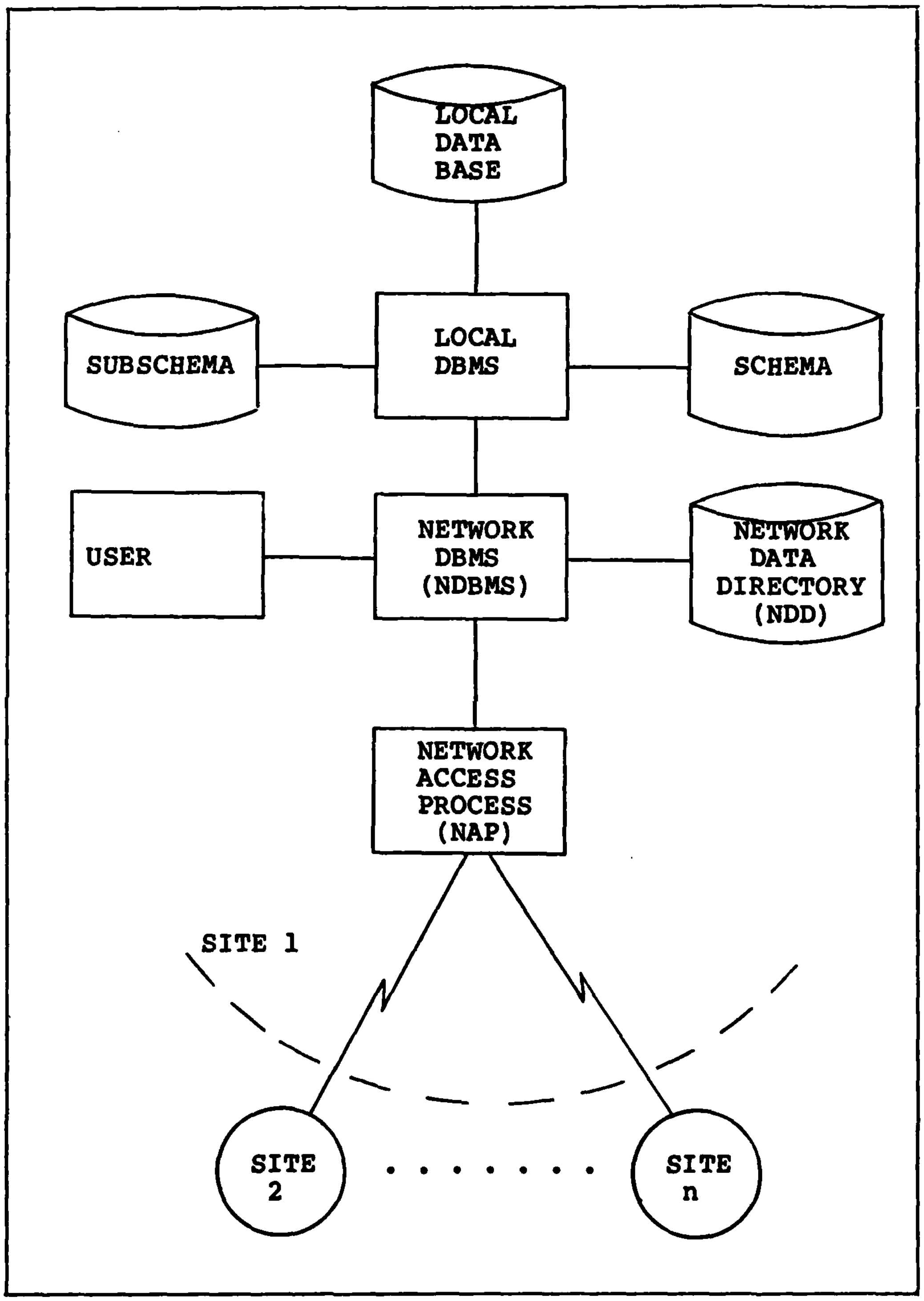


Figure 12 Components of a DDBMS at a Single Computer (Ref 3:64)

"Network" in their names. They are the Network Access Process (NAP), the Network Data Dictionary (NDD), and the Network Data Base Management System (NDBMS). These three components will be located at each computer in the network and are what constitute the DDBMS. The NDBMS components at each computer in the network are the controlling modules for operation of the DDBMS. Two of the NDBMS functions are the NAP interface and the NDD interface. The NAP and NDD components will be discussed first so that the interface functions of the NDBMS can be more easily understood.

Network Access Process

A Network Access Process (NAP) component is located at each computer in the network. It encompasses the communications hardware and software which links each computer to the rest of the distributed network. Figure 13 shows the NAP divided into two parts, the communication hardware and software area, and the NDBMS interface area.

The communication hardware and software area of the NAP handles functions such as message construction, message transmit timing, message acknowledgements, parity checking, checksum processing, and protocols. The standards to be implemented in this area for the DEL and specifically the LSINET are still being developed at this time (Ref 17:1-5). Current efforts in this area include

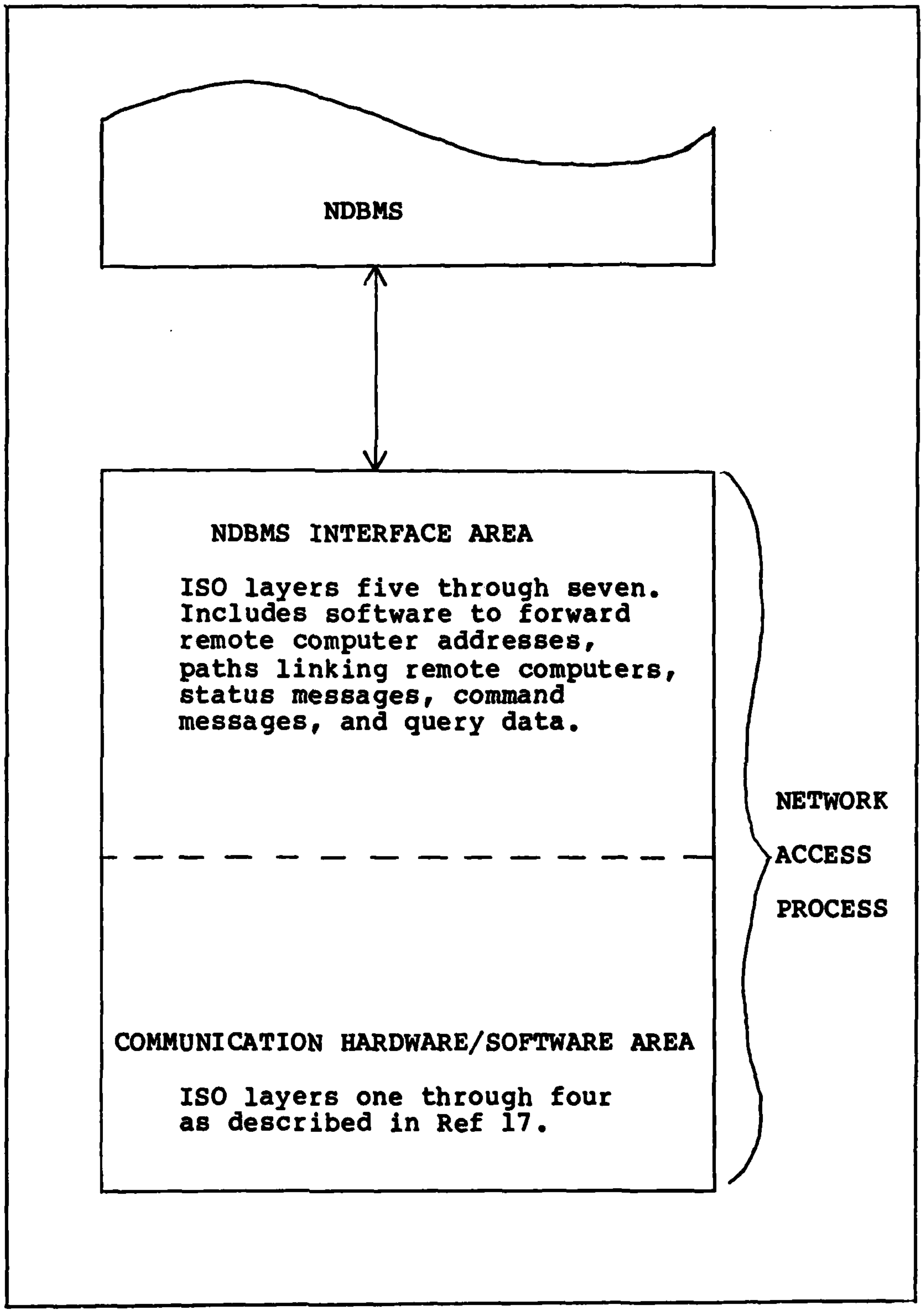

NDBMS
NDBMS INTERFACE AREA
ISO layers five through seven. Includes software to forward remote computer addresses, paths linking remote computers, status messages, command messages, and query data.
COMMUNICATION HARDWARE/SOFTWARE AREA
ISO layers one through four as described in Ref 17.
NETWORK ACCESS PROCESS

Figure 13 The Network Access Process Component

application of the International Standards Organization (ISO) seven layer protocol scheme. This particular ar[illegible] of the NAP encompasses only the first four layers of the ISO seven layer scheme. A detailed discussion of this subject in this design is beyond the scope of this thesis.

The NDBMS interface area of the NAP provides a "network description" to the NDBMS component. This description includes remote computer addresses and the paths linking them together. The status of all other computers in the distributed environment is also provided to the NDBMS. This area of the NAP also fowards query data and command messages between the NDBMS and other remote NAPs. This area encompasses the fifth through seventh layers of the ISO seven layer scheme.

In the AFIT DEL, the NAP software will be stored at each computer in the network. If a computer goes down, its NAP software also goes down. The NAP at each computer is responsible for detecting other computer failures and reporting them to their own NDBMS. In this environment the rest of the network and especially the distributed data base system can continue operating. This is an important design feature to provide for in any distributed data base system.

Network Data Directory

The Network Data Directory (NDD) is a data component. It is used by the Network Data Base Management System to determine where a query should be processed. At this point a universal data model design feature must be presented. In the heterogenous environment each local data base has its own data model. Thus for an NDD to be applicable, each local data model must be transformed into a common descriptive model. This will be known as a universal data model. The specific entities which make up all the local directory elements will be transformed into the universal data entities. This information will be stored in the NDD. The task of translation will be performed by each NDBMS to create its own local NDD. When a query is presented to the NDBMS, the query's specific entities are translated to the universal data model by the NDBMS and located using the local NDD. The location or locations of the data are then returned to the NDBMS for further processing.

The existance of a universal data model is usually considered a "weak link" in the design of a DDBMS. This is because of the complexity of transforming local directory elements from different DBMSs into universal data entities. However, the need for such a model has been confirmed by many other researchers (Ref 1:365). At this time, no entirely general translation scheme has been accomplished. The development of a "specialized" universal data model for the LSINET is required. This will be a significant part of the NDBMS design. The long range goal of this development effort would be to eventually establish a general purpose translation scheme.

So far only the concept of a local NDD at each computer in the network has been presented. The design of this DDB system will allow for replicated data. Therefore, the proposed NDD scheme has two other considerations which will now be presented. They are a Centralized NDD (CNDD) and an Extended Centralized NDD (ECNDD). The CNDD and the ECNDD directory structures are characteristic of a replicated data base distribution strategy.

A Centralized NDD (CNDD) will be established at one of the computers. This directory will be a union of all the local NDDs. It will be in the same universal data model described above. The computer which maintains this directory will not need a separate copy of its own local NDD because the local NDD will be incorperated into the CNDD.

The purpose of the CNDD is to provide remote computers with remote directory information. The CNDD will maintain the data entity names and all locations where each data entity exists. Since replicated data will be allowed in this system design, a data entity may be located at more than one computer. It is the responsibility of the NDBMS at all computers in the network to ensure that the CNDD is accurate.

At this time the computer which maintains the CNDD has been identified as being a key element to the distributed system design. If it should fail, the ability to locate new remote data by another computer would be prevented. This problem presents the issue of how a CNDD gets established in the first place. In this design each "suitable" NDBMS will be provided with the capability of becoming the site which has the CNDD. The process involves polling each computer in the network and building the CNDD. The term "suitable" implies a computer which has a predetermined minimum storage capacity available for a CNDD. Should the computer which maintains the CNDD fail, the next "suitable" computer will become the CNDD site. This involves a short period of time when CNDD information is unavailable. Should all "suitable" computers be unavailable, then the network continues with the information is has at hand. In some instances, acknowledging that a query cannot be processed at the current time must be accepted. This brings up the second additional NDD consideration, that of an Extended Centralized NDD.

This design will establish an Extended Centralized NDD (ECNDD) capability at each computer. This should provide faster response times to queries and reduce overall communication costs. The tradeoffs for this capability will include the following four costs. First, the cost of storing the extended directory. Second, the cost of storing the address of copied data in the centralized directory. Third, the cost of storing a more complicated update procedure. Fourth, the communication cost of updating other ECNDDs.

The extended feature will be implemented in the same format as the local NDD. The ECNDD will be established at any computer when a remote data entity location is required. When a local NDBMS obtains the location(s) of required remote data from the CNDD, the information is appended to the local NDD and becomes the ECNDD. Do not be confused by actual replicated data and the directory information indicating where the replicated data is located. The time to replicate data and where is a function of the NDBMS, which is discussed later.

Since the ECNDD could conceivably grow to the size of the CNDD, a decision needs to be made for when to purge an extended item from the ECNDD. For resolving this decision the ECNDD will be limited to "N" items. The value for N should be determined experimentally. Once the ECNDD is filled, the next required listing in the ECNDD will replace an existing extended item. The replacement process will use a Least Recently Used (LRU) algorithm. A counter field will be added to each ECNDD item. Each counter field is initially zero. Every reference by the NDBMS to an ECNDD item will increment the associated counter. The item with the lowest counter field value is replaced. An alternative would be a timestamp purge of an item. Each time an ECNDD item is referenced, a timestamp would be attached to it. If the ECNDD is filled, then the item that is replaced would be the one with the longest time since last referenced. Both the LRU and timestamp algorithms use extra storage and require additional software. However, it is felt that the LRU algorithm is a simpler and more flexible procedure for resolving the replacement issue.

Another issue related to the NDD is the problem of which computers have copies in their NDDs of other computer's data. For example, suppose computer A replicated a part of computer B's data. Let's refer to this replicated data as a "Parts" inventory. Computer A must report this informationto the CNDD. Now computer C requests "Parts" location(s) from the CNDD. Computer C will be told that it is located at computers A and B. Lets assume that computer C's query is completed and that time elaspes. Now computer A decides to purge "Parts". Computer A not only deletes the "Parts" listing from its own local NDD but also informs the CNDD to delete the fact that computer A has a copy of "Parts". At this time the CNDD and computer C's ECNDD are not the same. This design will implement a "Do Nothing" policy. First, lets examine the alternative. The CNDD computer would have to record not only which computers have replicated data but also which computers have requested remote data information. This would quickly turn into a major bookkeeping problem. The "Do Nothing" policy requires only that a local NDBMS comfirm that a data entity still exists at a remote computer before continuing to process a query. A single message transmitted by the local NDBMS and positively acknowledged by the remote NDBMS is required. Data locations obtained directly from the CNDD need not be confirmed since the CNDD is always kept current.

One final issue related to NDDs is what happens when a computer goes down. In this case the responsibility of establishing its local NDD rests with its own NDBMS. All ECNDD listings are lost and must be rebuilt from the CNDD as needed.

Network Data Base Management System

The network data base management system (NDBMS) is the final network related component of the DDBMS. An NDBMS will be located at each computer as shown in Figure 12. It has eight functions which are required because of the distributed environment. These eight functions are:

(1) User interface;

(2) Local DBMS interface;

(3) NAP interface;

(4) Local NDD interface;

(5) Query processing procedure;

(5a) Universal data model translator;

(5b) Repication process;

(5c) ECNDD process;

(5d) Universal query language translator;

(6) Concurrency control process;

(7) Backup process;

(8) CNDD process.

Figure 14 shows the interconnections between each of these functions. The structure design described in Figure 14

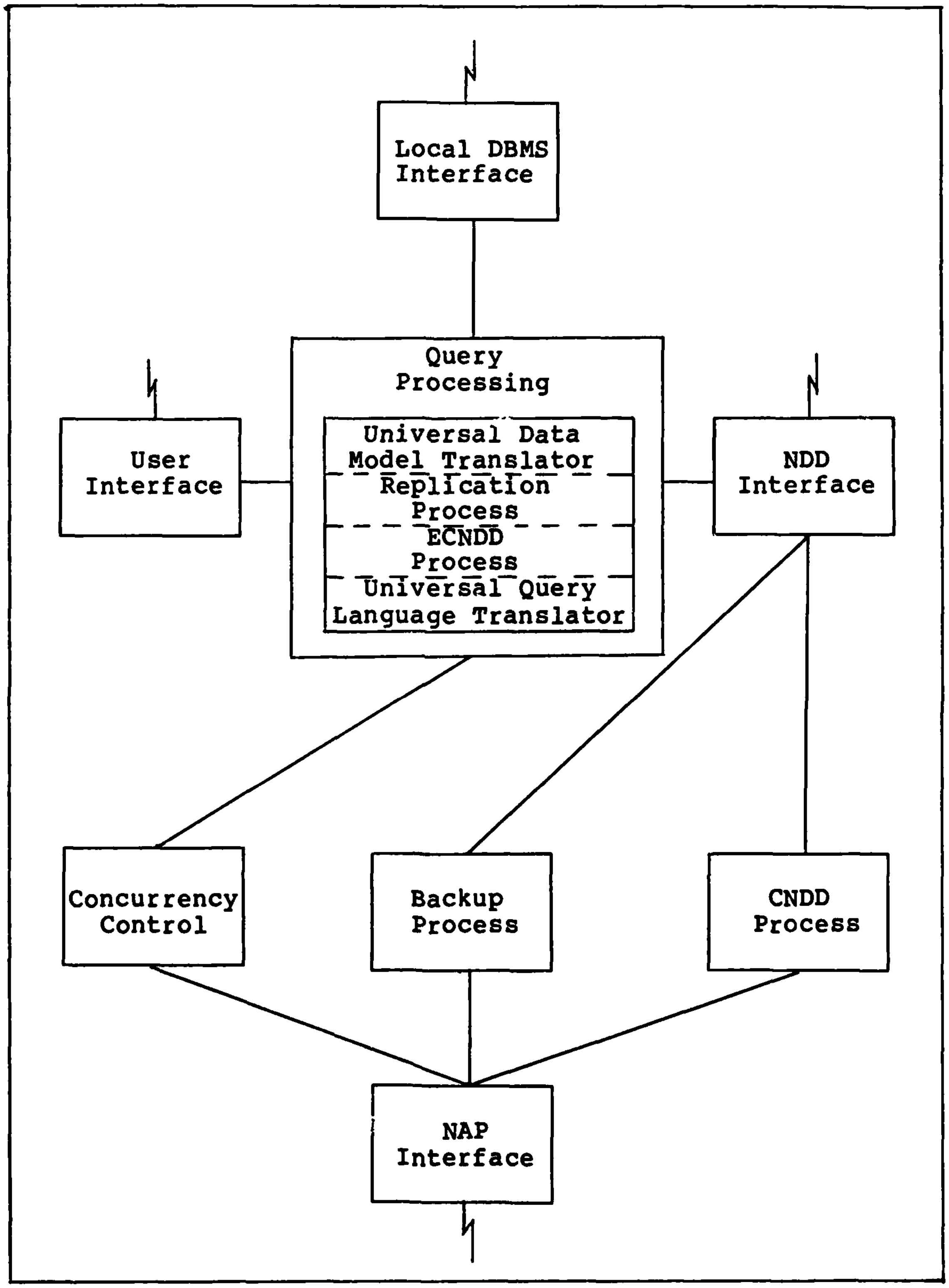


Figure 14 The NDBMS Structure

attempts to decentralize control so that the network may continue to operate in the event of computer failures at individual sites. Figure 15 shows the general operating strategy for processing a query.

The function of the NDBMS is to provide an interface between the user and the data base. When the NDBMS processes a request for data, it accepts the query in the user query language of the local DBMS. The specific entities (the names of data items in the query) of the request are translated into universal data entities. These entities are compared with the information found in the NDD. If the request requires only local data then the original request is passed to the local DBMS for processing. No translation of the request is made except for those requiring NDD comparisions. The local DBMS processes the request and the result is returned back to the NDBMS. The NDBMS then provides the result to the user. The NDBMS is involved in controlling this processing for concurrency control reasons which will become evident later. If the request requires remote data, then special processing takes place.

To process remote data the NDBMS will have a universal query language translation capability. Although a general universal query language does not exist at this time, a specialized query language can be adapted which will use the NDD entities for reference. When the NDBMS requires remote data manipulation, a query is built which can be

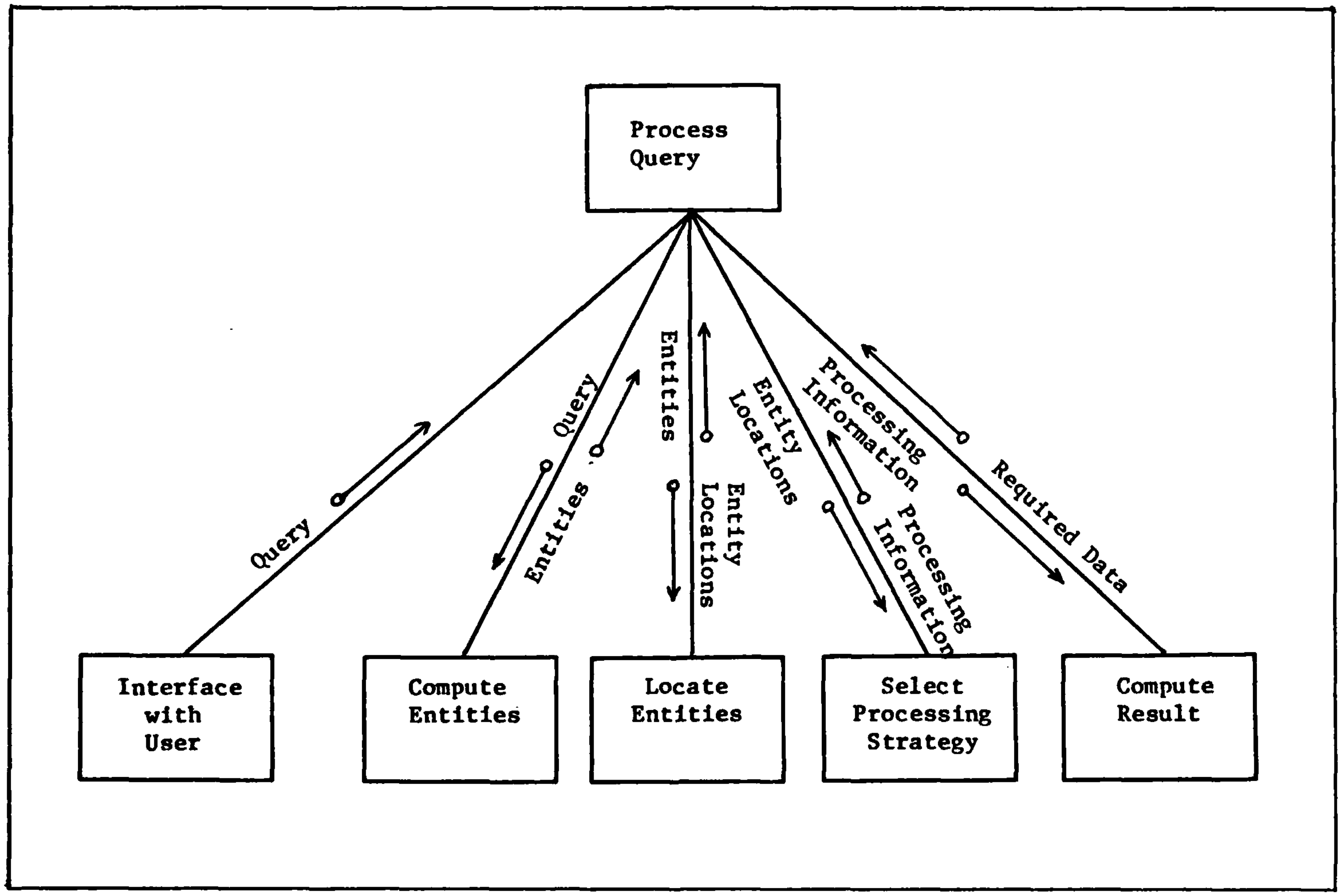


Figure 15 Operating Strategy for Processing a Query

forwarded to the remote computer that has the necessary data. The query is then decomposed at the remote computer's NDBMS into its own query language for processing. The resulting data is then returned to the original computer's work area for further processing.

Up to this point, nothing has been mentioned about optimizing remote request strategies. This is an important function of the NDBMS. To accomplish this let's return to the discussion of the Network Access Process (NAP). The NAP provides the NDBMS with remote computer addresses and paths linking them together. The NDBMS obtains all the locations of remote data from the NDD and then using the NAP data computes the optimum processing site(s). The processing of requests requiring remote data is considered for two types of requests, simple and compound requests. A simple remote request is one for which all the data is located at one remote computer. In this case the NDBMS determines to which computer to forward the request for processing. The request is processed completely at that remote computer and the result returned. A compound request involves decomposing the request into a set of local and remote requests or multiple remote requests. In this case the query processing strategy developed by Eugene Wong (Ref 22:50-68) will be used by the NDBMS for selecting the processing method. Recall this is a recursive technique which finds an optimum processing strategy by minimizing data movement.

The query processing strategy also will maintain a table to indicate which remote data entities are used and how often. The purpose of this function is to determine if and when to replicate remote data based on usage. The table referenced above will be physically located in the ECNDD. This uses the fact that if remote data is being considered then that particuliar computer has a nonempty ECNDD. The ECNDD described earlier identified a counter field will be added to each ECNDD item. Each counter field is initially zero. Every reference by the NDBMS to an ECNDD item will increment the associated counter. If the counter reaches "M" (an experimentally determined value) before the item is purged from the ECNDD then the data for this item is replicated at the local computer if possible. This process includes three maintenance functions. First, removing the item from the ECNDD. Second, adding the item to the local NDD. Third, informing the CNDD of the replication. The automated function of replication of remote data into a local data base is extremely complicated. However, with the establishment of a universal data model and translation capability, the replication function should be obtainable.

Another function of the NDBMS is concurrency control. When replicated data is allowed, as in this design, the monitoring of updates is important. The majority voting algorithm will be used for this purpose. This algorithm is selected for two reasons. First, and most importantly, it has been shown to work correctly. That is, all copies of the replicated data converge to the same value and internal data base consistency is preserved. Second, the algorithm is deadlock free.

The NDBMS will provide a backup and initialization function. When a computer fails, all other computers in the network are informed by their associated NAPs. When the computer which maintains the CNDD is informed, it checks its CNDD for replicated data entities which were involved with the lost computer. If a replicated entity is found then a message is transmitted to the remote computer involved. The message commands the remote computer to begin maintaining an audit trail for the replicated data. This audit trail will be maintained for 24 hours (arbitrarily choosen) or until a predetermined number of enteries in the audit trail are made. This cutoff mechanism is used to decide whether implementing an audit trail recovery is more cost beneficial than a replication effort of the data.

At the time the down computer comes active again the local DBMS will recover its own local data base. When this is completed the NDBMS creates the local NDD. The local NDD is then compared to the CNDD to locate any replicated data. The decision is then made by the recovering computer's NDBMS as to whether it will update using the other computer's audit trail or replicating the data.

V. NETWORK DATA DIRECTORY SYSTEM DESIGN

Introduction

The design of the directory management system component consists of three parts. The first part is the logical Network Data Directory (NDD) design. The second part describes software procedures used by the Network Data Base Management System (NDBMS) for processing queries in the distributed environment. The third part is the physical network data directory structure requirements and description. The physical structure described in this chapter uses Pascal declarative structures for presenting detailed parts of the software components.

Logical Network Data Directory Design

The purpose of accessing the NDD is to locate data in the distributed environment. This function should be as fast as possible. It is done by comparing entity names found in queries with entity names found in the NDD. Entities are names of data items referenced in queries. The following example shows how Relational, Hierarchical, and Network DBMS data items found in queries might be translated into entity names. Suppose the data items shown in Figure 16 represent the data base. In the Relational form, a table name is specified along with its attributes, as in Figure 17a. In che Hierarchical form, a root node with its attributes is superior to subordinate

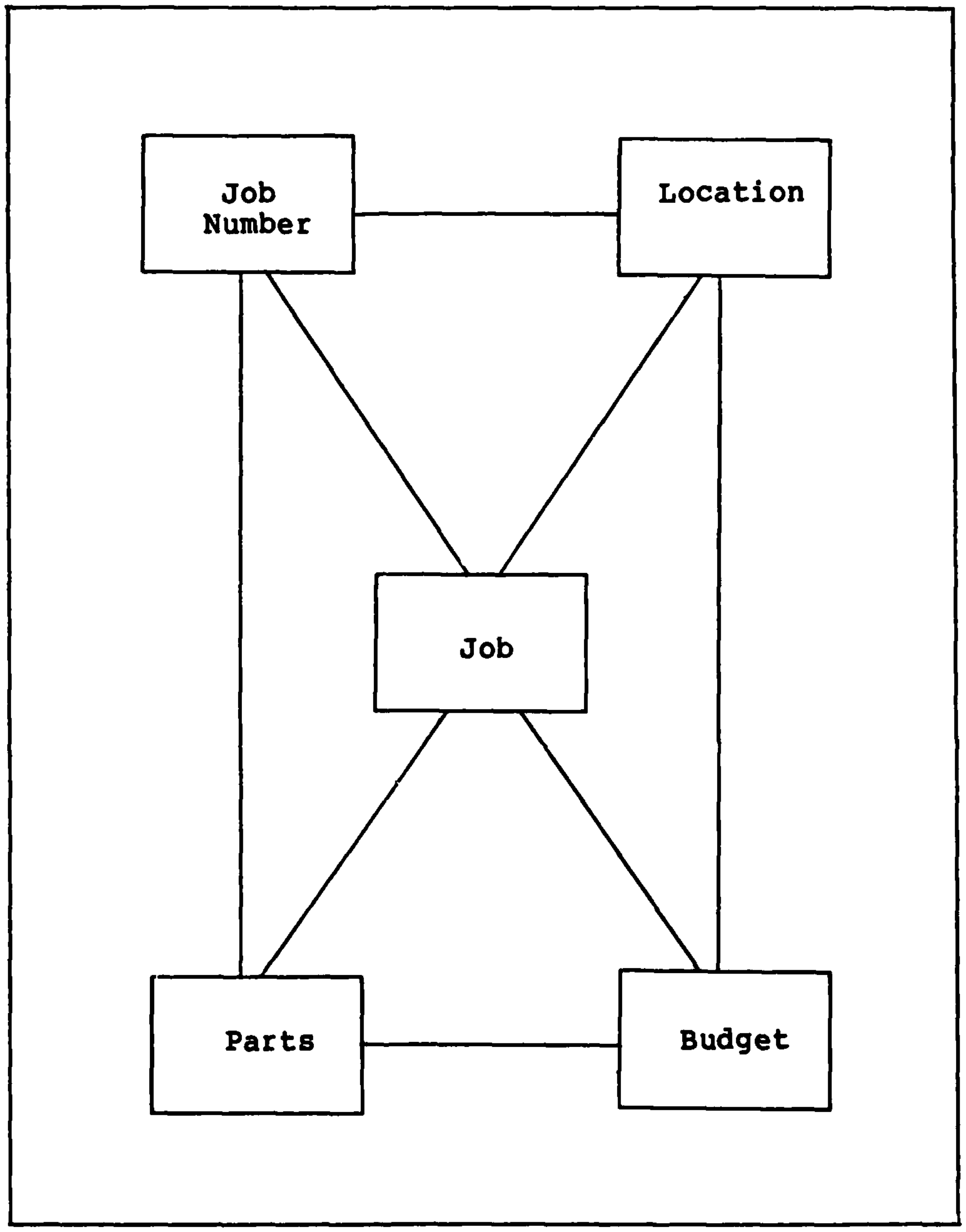


Figure 16 Example Data Items in a Data Base

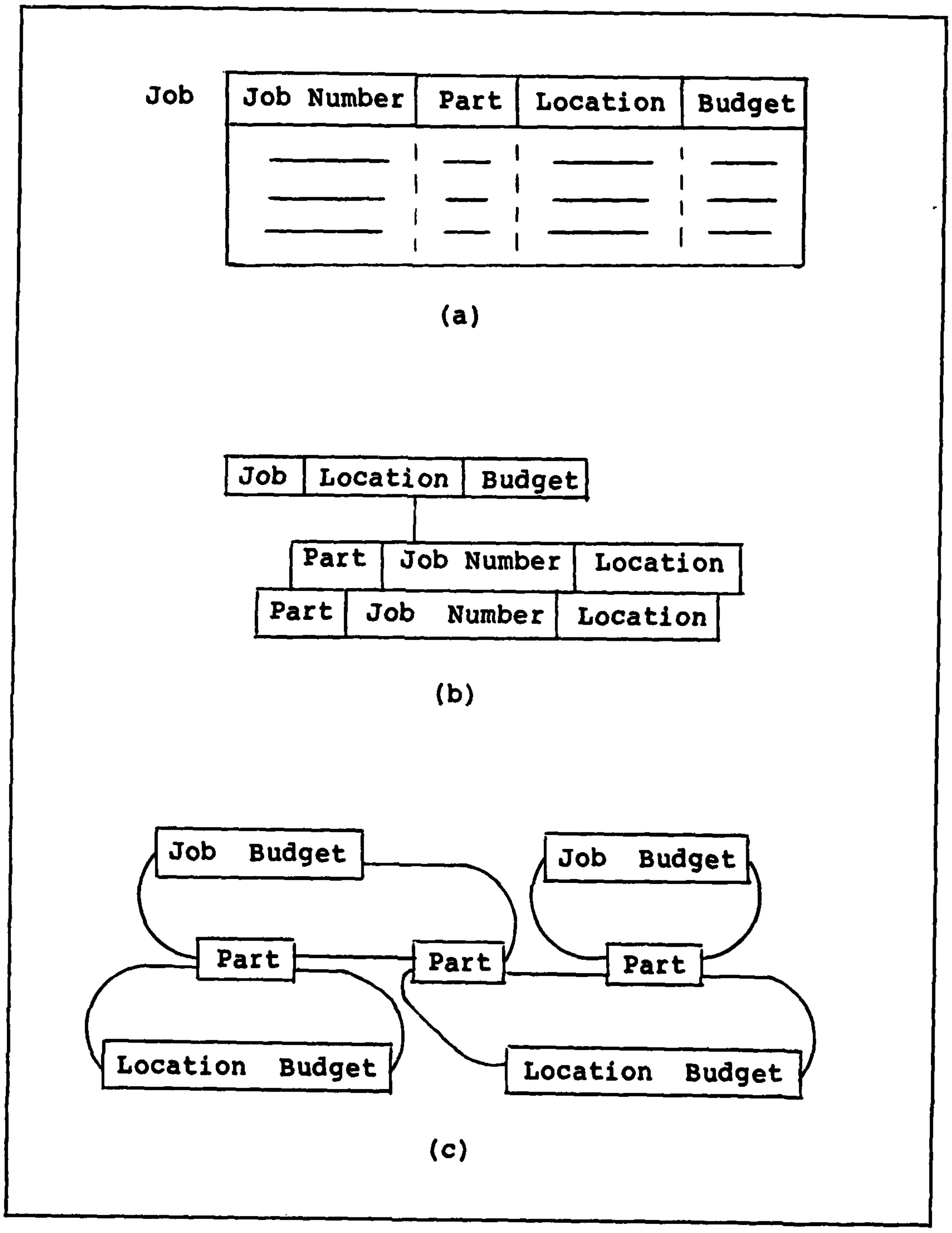


Figure 17 (a) Example Relational Form
(b) Example Hierarchical Form
(c) Example Network Form

nodes, as in figure 17b. In the network form a record with its attributes is linked with other records as in Figure 17c. The names of tables, nodes, and records are all considered data items and would be translated into entity names. A query such as, "Find job numbers for jobs that require part xyz," would require data item "job" be translated into entity "job". Since "job number" and "part" are only attributes in this example, they are not translated. Locating the proper computer which has the "job" data should be possible in all three DBMS forms.

The entity names will be stored in three types of data structures. The structures will be identified as follows:

(1) Local Network Data Directory (LNDD);

(2) Centralized Network Data Directory (CNDD);

(3) Extended Centralized Network Data Directory (ECNDD).

Each data structure will be specialized for its intended use so that searching it can be accomplished in the shortest possible time. The processes using these data structures are described in the next three sections.

Local Network Data Directory Processes

The first step involving the directory component is done by the NDBMS universal data model translator. As described earlier, it transforms all query items and local data base items into a common entity form used throughout the distributed data base. Thus, a procedure must be available to create the LNDD from the entity names presented to it by the translator. This procedure, CREATELNDD, is executed before any queries are entered into the system. Another procedure is required to forward this information and any new or deleted information concerning the LNDD, to the computer which maintains the CNDD. This procedure will be called UPDATECNDD. Figure 18 shows the relationships of these procedures with other procedures required for the NDD interface.

The query processor in the NDBMS uses the directory component in a number of ways. The NDBMS at each computer will have a procedure which will accept a query and, using the translator, compare the entities with those stored in the LNDD. This procedure will be called CHECKLNDD. If all entities are found to be available at the local computer then the query is forwarded to a local DBMS interface function for processing. If an entity is not found in the LNDD then the ECNDD is checked. This procedure, called CHECKECNDD, returns either a "null" indicating Not Found or a list of remote computer addresses where the entity is located. If a list of remote computer addresses is returned then the query processor uses this information for processing. If a "null" is returned then a message is coordinated through the Network Access Process (NAP) to ask the computer with the CNDD where to find the entity.

Extended Centralized Network Data Processes

The creation of the ECNDD is discussed in this section.

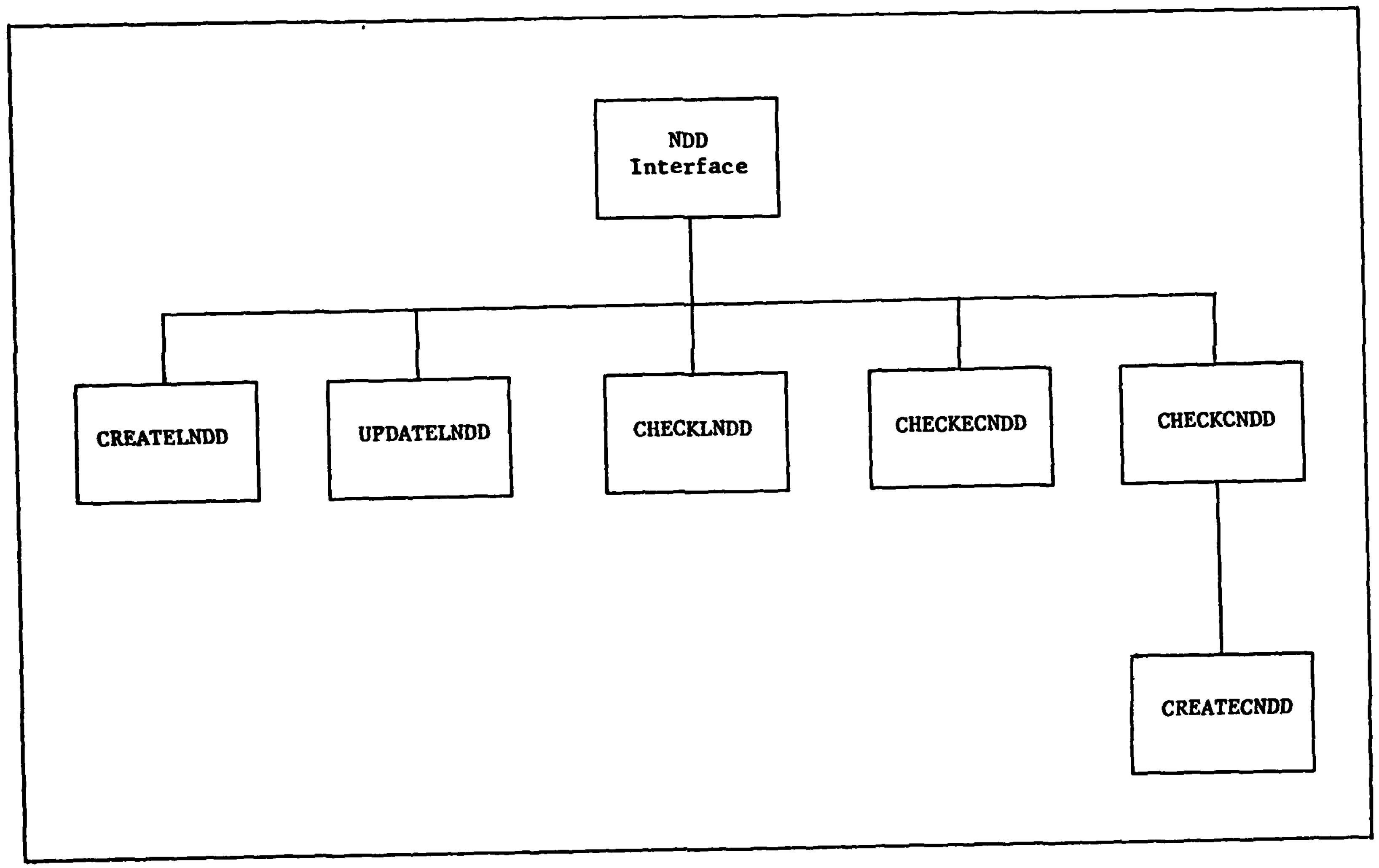


Figure 18 Procedures Associated with the NDD Interface

Whenever CHECKECNDD returns "null", the CNDD will be contacted. A procedure which checks the CNDD, CHECKCNDD, will provide either a "null" of its own or a list of computer addresses indicating where the entity is located. A "null" from the CNDD means that the entity is not in the distributed data base, thus the query involving that entity cannot be processed. A list of computer addresses returned by the CNDD is used to create the ECNDD at the local computer. The entity name and computer addresses are stored in the ECNDD by a procedure called CREATEECNDD. At that time a counter field for that entity is set to 1. Therefore, locating the same remote data a second time can be done locally. Whenever CHECKECNDD successfully locates an entity for the query processor, that entity's counter is incremented. When the counter reaches a predetermined value, that entity is considered for replication at the local computer. When replication is performed, three maintenance functions involving the data directory are required. First, the entity name is removed from the ECNDD. Second, the entity name is added to the LNDD. Third, the CNDD is notified of the change.

To prevent the ECNDD from growing to the size of the CNDD, the ECNDD will be limited to "N" items. Once the ECNDD is filled, the next entity to be stored in the ECNDD will replace an existing one by using a Least Resently Used (LRU) algorithm. The counter field associated with each extended entity will be used for this purpose. The entity with the lowest counter value is replaced.

Whenever the query processor receives a remote computer address from the ECNDD, it transmits a message to confirm that the entity is still at the remote computer. This is part of the "Do Nothing" policy described in Chapter 4, Network Data Dictionary section. This policy dealt with keeping track of which computers have listings of remote data locations. This policy is required due to the fact that only the CNDD is kept current in regards tc replicated data locations.

Centralized Network Data Directory Processes

One of the computers in the distributed network must maintain the CNDD. The processes associated with the CNDD functions will be available to all computers in the network. That is, the procedures for creation of the CNDD and its responses to remote data locating requests will be incorporated into all computers which have the storage space available to maintain a CNDD. A predetermined takeover order of the CNDD function will be available in the event the computer which currently maintains the CNDD fails. The CNDD processes will include:

(1) Building of the CNDD from remote LNDDs;

(2) Maintaining accurate remote data locations by receiving messages from remote computers;

(3) Responding to remote data location requests.

In one respect, the query processing procedure at the

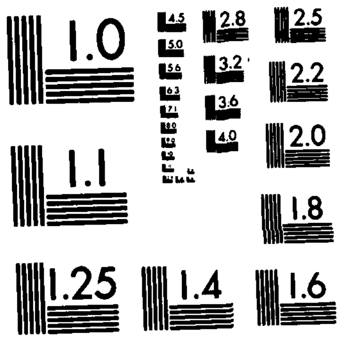
AD-A124 821 DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT D..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. E F IMKER 2/2

UNCLASSIFIED DEC 82 AFIT/GCS/EE/82D-21 F/G 9/2 NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CNDD computer is faster because the data entity locations required for a query are readily available. However, because of interrupts by other computers and the length of the CNDD itself, the actual processing time may be longer. The need for a separate LNDD at the CNDD computer should be determined experimentally based on query processing delays associated with maintaining the CNDD.

Other Network Data Directory Processes

The emphasis of the previous three sections has been the design of data directories and their operating processes. Nearly every component of the NDBMS will require use of one or more of the procedures described earlier. For example, the query processing function requires data directory access for update processing. Whenever data updates are required, all replicated data locations must be found. Another NDBMS function requiring data directory access is the backup process. In other words, as each of these other processes are designed, their interface to the data directory must formulated to fit their requirements. The data directory structures provided should make the data locating processes in these other NDBMS functions easy.

Physical Network Data Directory Structure

This section provides for the design of the three data structures that will be used by NDBMS directory software processes. The structures, described in order of complexity, are the LNDD, CNDD, AND ECNDD.

The LNDD will be a structure which contains all universal data model entity names that are stored in the local data base. Its size will be determined by the number of entities stored in the local data base. Let's assume the local data base has "k" entity names. It must be searchable in the shortest possible time since it will be referenced by every query submitted to the NDBMS. Additionally, it must be able to handle additions or deletions routinely. The solution presented here is based on the binary AVL tree data structure. The nice feature of an AVL tree is that the tree structure remains balanced with respect to the heights of subtrees (Ref 15:442-456). The AVL definition also requires that all subtrees be height balanced. As a result, dynamic retrievals can be performed in O(log k) worst case time, where "k" is the number of local data base entity names. In the case of a binary search tree, a possible alternative, the worst case time is O(k). Additionally, a new node in the tree can be entered or deleted in O(log k) worst case time. Figure 19 illustrates this structure. Its declarative structure in Pascal is:

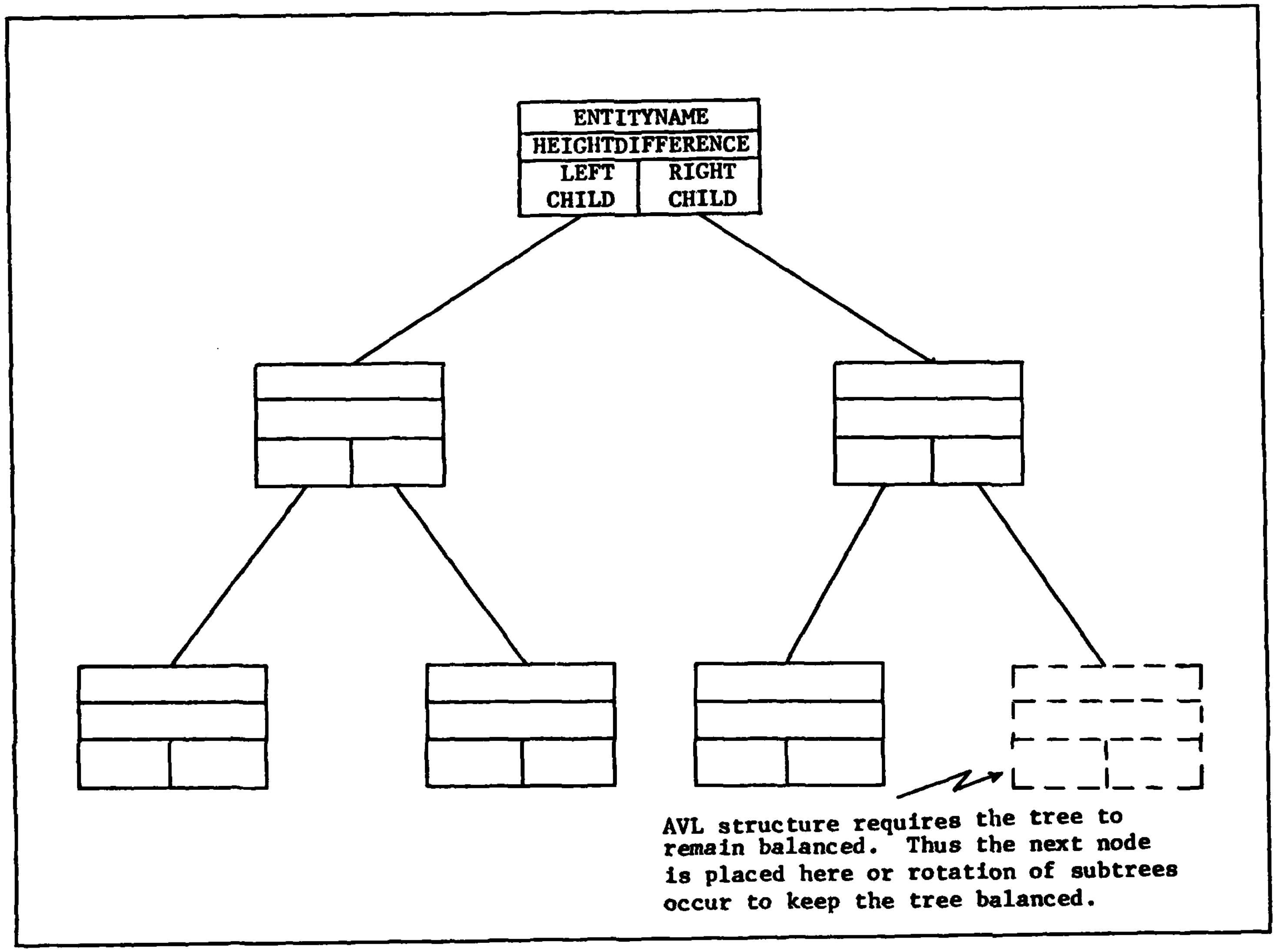

ENTITYNAME
HEIGHTDIFFERENCE
LEFT CHILD
RIGHT CHILD
AVL structure requires the tree to remain balanced. Thus the next node is placed here or rotation of subtrees occur to keep the tree balanced.

Figure 19 LNDD Data Structure

```
TYPE
  LNDD = ↑LNDDRECORD;
  LNDDRECORD = RECORD
                 ENTITYNAME : ALFA;
                 HEIGHTDIFFERENCE : INTEGER;
                 LEFTCHILD : LNDD;
                 RIGHTCHILD : LNDD
               END;
VAR
  LNDDTREE : LNDD.
```

The Centralized Network Data Directory (CNDD) will be a structure which contains all entity names used throughout the entire distributed data base. However, the CNDD must also provide the locations of all replicated data when required. For speed reasons, the CNDD will also be a binary AVL tree construct similiar to the LNDD structure. An additional secondary linked list record is added to the declarative structure for indicating replicated data locations. Figure 20 illustrates this structure. The declarative structure in Pascal is:

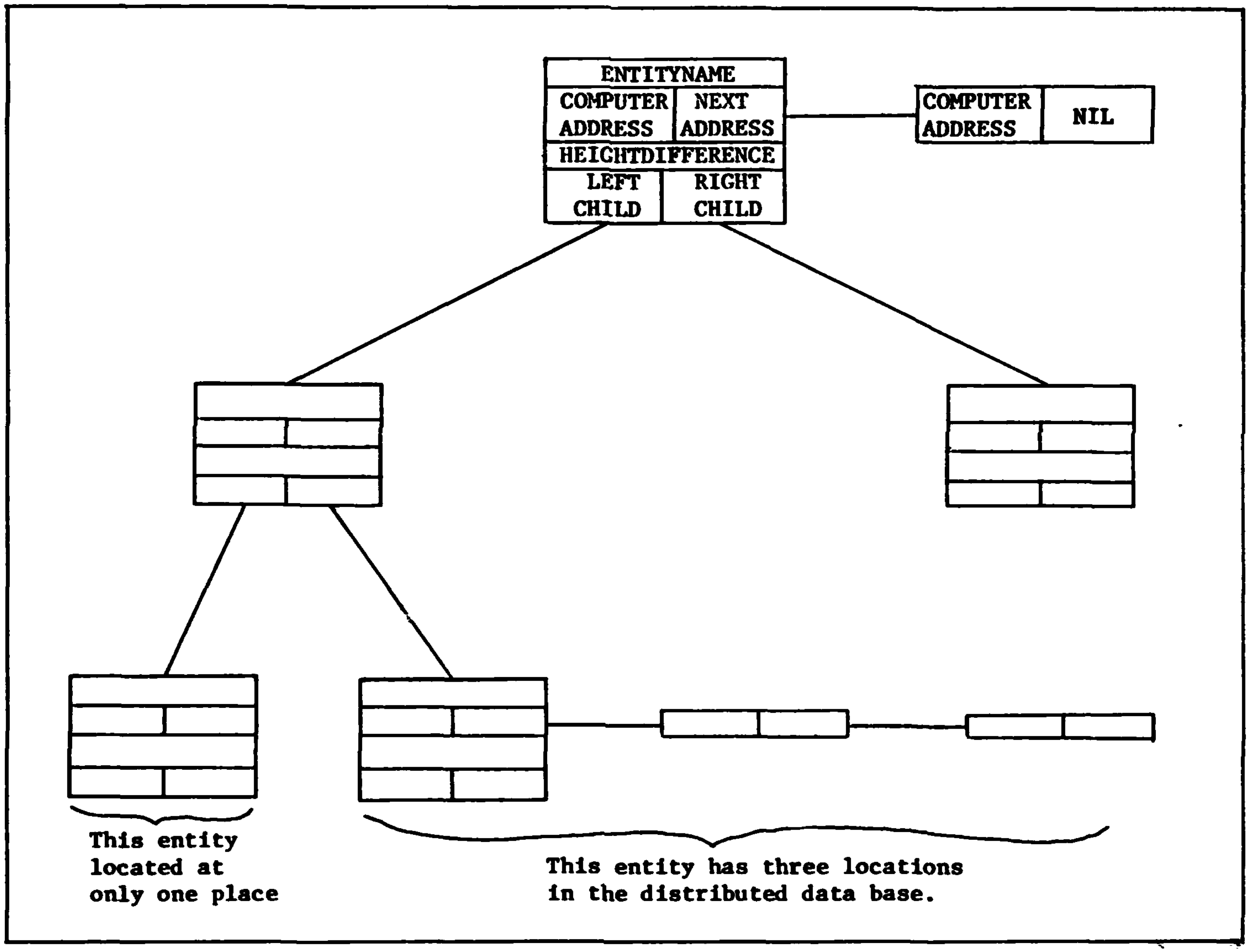

ENTITYNAME
COMPUTER ADDRESS
NEXT ADDRESS
HEIGHTDIFFERENCE
LEFT CHILD
RIGHT CHILD
COMPUTER ADDRESS
NIL
This entity located at only one place
This entity has three locations in the distributed data base.

Figure 20 CNDD Data Structure

```
TYPE
  LOCATION = ↑LOCATIONRECORD;
  LOCATIONRECORD = RECORD
                     COMPUTERADDRESS : INTEGER;
                     NEXTADDRESS : LOCATION
                   END;
  CNDD = ↑CNDDRECORD;
  CNDDRECORD = RECORD
                 ENTITYNAME : ALFA;
                 REPLICATIONS : LOCATION;
                 HEIGHTDIFFERENCE : INTEGER;
                 LEFTCHILD : CNDD;
                 RIGHTCHILD : CNDD
               END;
VAR
  CNDDTREE : CNDD.
```

The Extended Centralized Network Data Directory (ECNDD) will be a structure which contains remote data locations for the local computer's NDBMS. It's length will be considerably shorter than that of the LNDD. It will be consulted after failing to locate an entity name in the LNDD. Since it is relatively short, it will be a linked list with no sorted pattern. It consists of an entity name, a counter and a secondary linked list. The counter is used for determining when to replicate data. The secondary linked list is used for indicating replicated data locations just as in the CNDD structure. Figure 21 illustrates this structure. The declarative structure in Pascal is:

```
TYPE
  ECNDD = ↑ECNDDRECORD;
  ECNDDRECORD = RECORD
                  ENTITYNAME : ALFA;
                  REPLICATIONS : LOCATION;
                  COUNTER : INTEGER;
                  NEXTENTITY : ECNDD
                END;
VAR
  ECNDDLIST : ECNDD.
```

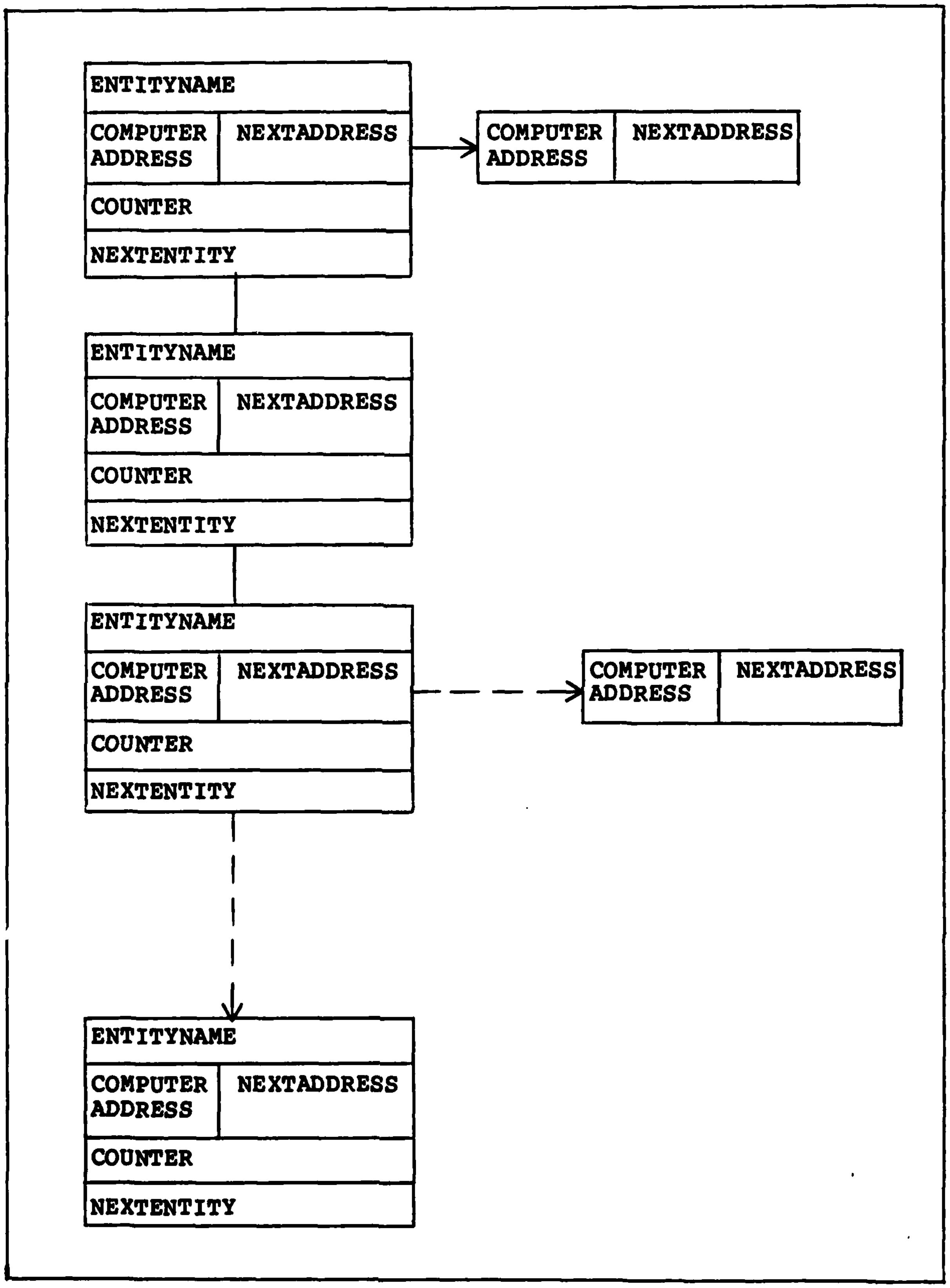


Figure 21 ECNDD Data Structure

VI. CONCLUSIONS AND RECOMMENDATIONS

Overview

In this thesis investigation, a top level design of a Distributed Data Base Management System (DDBMS) was attempted. The research emphasis was in two areas. First, an extensive literature search of existing and proposed DDBMSs was performed. Second, the top level design of a DDBMS which could be implemented at AFIT was presented.

The background investigation pointed out that the complexity of a distributed data base (DDB) system depends on the architecture choosen. Since the AFIT DEL has such a variety of different DBMS's available, a heterogenous architecture was selected for study. This particular architecture is also the newest area of study in the DDB world and thus could provide significant contributions to future research efforts.

The only data allocation method acceptable in a true DDB environment is one which includes replicated data capabilities. Replicated data allows the query processing optimization process to select the strategy that is most appropriate for individual queries. In other words, speed in providing answers to queries from as large a data base that can possibly be created is the goal of a DDB system.

It was determined in the literature research that there is no best solution for concurrency control. There are numerous methodologies being studied at this time, each with known advantages and disadvantages, depending on their environment.

The data directory approach designed in this thesis selected the centralized directory system with extended centralized capability. The design is relatively simple but it is enough to provide the local computer's Network Data Base Management System all the information required for processing queries. It was also designed to distribute the control of the Centralized Network Data Directory in case of computer failures.

Decentralizing the data management process is one of the most inportant factors in designing a DDBMS. The network should not come to a halt just because one or two computers go down. This coincides with the conclusion that reliability and recovery mechanisms are essential in the design of DDBMSs.

Recommendations

Many areas of the design have to be developed in more detail. Developing a universal data model and query language is the most important. The translations required from one local DBMS to another are complicated and have not yet been fully realized.

Follow-on thesis efforts should also concentrate on devel[illegible]ment of the query processing optimization strategy as well as the concurrency control mechanism. Each of these areas by themselves could constitute an entire thesis effort.

It is difficult to perceive any one individual putting all the different development packages together and making it work. Implementation would probably be best accomplished by a team of individuals. Initial implementation of an operational DDBMS should remain on the LSINET until all the components are working properly. Complicating the problem by including different hardware in the early stages will only delay the implementation and make debugging much more difficult. A specific top down design is needed. Detailed design plans using a software engineering tool such as SADT are needed. As further design specifications are made, design reviews will become essential.

Final Comment

DDBMSs are often only talked about and rarely implemented. This thesis was an attempt at designing a DDBMS that could be implemented in the AFIT DEL. Though only a small portion of the design effort was accomplished, a starting point for future work has been established. Chapter 2, a background study, was a major portion of this thesis. The top level design presented is a combination of good ideas that have been uncovered during the research. Since only the NDD component was developed in the lower level design stage, there is much flexibility for alteration of the rest of the design. As time progresses some of the solutions to questions left unanswered will undoubtably surface in DDB literature. It is hoped that this thesis will contribute in some way to a more advanced understanding of distributed data base management systems.

Bibliography

1. Adiba, M., et al. "Issues in Distributed Data Base Management Systems: A Technical Overview," Tutorial: Centralized and Distributed Data Base Systems, 359-380. New York: IEEE Computer Society, 1979.

2. Bernstein, Rothnie and Shipman. Tutorial: Distributed Data Base Management. New York: IEEE Computer Society, 1978.

3. Bray, Olin H. Distributed Database Management Systems. Lexington: D. C. Heath and Company, 1982.

4. Champine, G. A. "Six Approaches to Distributed Databases," Datamation, 23: 69-72 (May 1977).

5. Champine, George A. Distributed Computer Systems Impact on Management, Design, and Analysis. New York: North-Holland Publishing Company, 1980.

6. Chu, Wesley W. "Performance of File Directory Systems for Data Bases in Star and Distributed Networks," Tutorial: Centralized and Distributed Data Base Systems, 442-457. New York: IEEE Computer Society, 1979.

7. Date, C. J. An Introduction to Database Systems (Third Edition). Reading: Addison-Wesley Publishing Company, 1981.

8. Donaldson, Hamish. Designing a Distributed Processing System. New York: Halsted Press, 1979.

9. Fonden, Capt Robert W. "Design and Implementation of a Backend Multiple-Processor Relational Data Base Computer System," Unpublished Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

10. Geist, Capt John W. "Development of the Digital Engineering Laboratory Computer Network: Host-Node/Host-Host Protocols," Unpublished Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

11. Hammer, Michael and David Shipman. "An Overview of Reliability Mechanisms for a Distributed Data Base System," Tutorial: Centralized and Distributed Data Base Systems, 465-475. New York: IEEE Computer Society, 1979.

12. Hartrum, Thomas C., Assistant Professor of Electrical Engineering. Lecture materials on Distributed Data Base Systems presented in EE6.46, Computer Data Base Systems. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, March 1982.

13. Henver, Alan R. and Bing S. Yao. "Query Processing on a Distributed Database," Tutorial: Distributed Data Base Management, 69-85. New York: IEEE Computer Society, 1978.

14. Hobart, Capt William C. Jr. "Design of a Local Computer Network for the AFIT Digital Engineering Laboratory," Unpublished Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

15. Horowitz, Ellis and Sartaj Sahni. Fundamentals of Data Structures. Potomac: Computer Science Press, Inc., 1976.

16. Katzan, Harry Jr. An Introduction to Distributed Data Processing. New York: Petrocelli Books, Inc., 1978.

17. Lamont, G. B. "AFIT/ENG Digital Engineering Laboratory Computer Network and Applicability to AFIT Local Area Network," Unpublished letter to AFIT/ACD (Lt Col Wassem). School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, August 19, 1982.

18. Menasce, Daniel A. and Richard R. Muntz. "Locking and Deadlock Detection in Distributed Databases," Tutorial: Distributed Data Base Management, 95-112. New York: IEEE Computer Society, 1978.

19. Peebles, Richard and Eric Manning. "System Architecture for Distributed Data Management," Tutorial: Centralized and Distributed Data Base Systems, 351-357. New York: IEEE Computer Society, 1979.

20. Roth, Lt Mark A. "The Design and Implementation of a Pedagogical Relational Database System," Unpublished Thesis. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1979.

21. Thomas, Robert H. "A Solution to the Concurrency Control Problem for Multiple Copy Data Bases," Tutorial: Distributed Data Base Management, 88-94. New York: IEEE Computer Society, 1978.

22. Wong, Eugene. "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," Tutorial: Distributed Data Base Management, 50-68. New York: IEEE Computer Society, 1978.

APPENDIX A

Publication Article

Report on

DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT DIGITAL ENGINEERING LABORATORY

Abstract

A distributed data base management system (DDBMS) was designed with the goal of providing the AFIT School of Engineering a research tool. The objective was to use state-of-the-art knowledge in a design that would provide an experimental testbed to further advance DDBMS knowledge.

Toward this goal, an extensive investigation was made into distributed data base systems. Numerous alternatives were presented in the areas of configurations and architecture, data allocation, directory management, query processing, concurrency control, reliability, integrity and security. This background discussion includes advantages and disadvantages of the alternatives.

A top level system design was presented which includes replication of data, a universal data model and query language, a centralized and extended centralized directory system and the majority voting concurrency control algorithm. Due to the complexity and size of the

development effort, a detailed design of only the directory system was made in this research effort. Follow-on development efforts over the next few years will be required to complete this research project.

Statement of the Problem

The Air Force Institute of Technology (AFIT) Digital Engineering Laboratory (DEL) is currently involved in several areas of data base research. A distributed data base system sharing stored data would provide a significant upgrade of the existing facility. In a few years, memory limitations would be reached on individual computer systems. At that time an operational distributed data base would be valuable. This upgrade would be intended initially as a research tool for the DEL. Eventually the distributed data base would become a pedagogical tool for students here at AFIT as well as an operational tool.

This research effort was concerned with the design of a distributed data base management system (DDBMS) for the DEL. The specific purpose of this research was to provide a top level design of a DDBMS system that meets the requirements of the DEL. Desirable alternatives of the issues were discussed and provided for in the design.

An extensive literature search was made to analyze operational distributed data base management systems (DDBMS). The objective here was to avoid a "reinvent the wheel" type of approach. Unfortunately, most operational

DDBMSs are designed for a unique environment; however, basic concepts are the same. A top level system design for a DDBMS was made. Additionally, the design of a directory management system component was made.

Background Summary

There are many types of DDB systems. Each has been developed for the environment which it will be used. This research pointed out that the architecture strategy is significant in determining the complexity of the system. There are three basic approaches to DDB architecture. They are, from simpler to more complex, integrated, homogenous, and heterogenous.

The three data allocation methods generally used are centralized, partitioned, and replicated. Query processing optimization can be most efficient in a replicated scheme. However, the recommended approach depends on specificity, update frequency, and the amount of data.

The three directory management approaches are centralized, local, and distributed. Choosing the method for any particular network depends on the computer network topology, communication cost, storage cost, directory query rate, and directory update rate.

Query processing is the key to speedy data retrieval. Optimization of query processing requires parallel data activities such as data reductions and data movement.

Improvements to an Initial Feasible Solution are made by recursively finding data movements that reduce the size of relations, thereby reducing the time for subsequent data movements.

The concurrency control issue addresses the problems of updating a multiple copy data base and synchronization of transactions. Three solutions to these issues are a broadcast mechanism, a timestamp mechanism, and a majority voting algorithm. One method can not be called better than the others because the network environment provides different advantages and disadvantages for each.

Reliability in a DDB system concerns computer failures and subsequent recovery procedures. In a replicated data environment, the problem of data updates missed by a failed computer become important. In some cases an audit trail is used to update the data during recovery. In other cases the data is copied from current replicated data. Detection of failed computers is essential, therefore reliability algorithms usually include a network monitoring activity.

Integrity in a DDB system refers to the accuracy, validity, consistency, and availability of the data. Each of these factors contribute to the degree of integrity in a DDB system. Generally, higher (better) degrees of integrity require higher costs to implement.

Security problems in a DDB system are usually extensions of existing methods used in centralized systems. These methods include access decisions, access authorizations and data encryption.

Components of the DDBMS

The design of this distributed data base system management system (DDBMS) is described in this section. For pedagogical and research purposes, strong emphasis was placed on functional modularity so that some components can be changed without affecting others. The components that would be necessary in a heterogenous architecture environment are identified. A heterogenous environment has been deliberately selected so that the most general purpose DDBMS can be developed. A general purpose DDBMS has not been previously developed although many computer laboratories are actively working on the subject.

To describe the components of this distributed data base system a single computer will be examined. This computer may implement on the local level a relational, hierarchical, or network data base. The three main components are the Network Access Process (NAP), the Network Data Dictionary (NDD), and the Network Data Base Management System (NDBMS). These three components will be located at each computer in the network and are what constitute the DDBMS. The NDBMS components at each computer in the network are the controlling modules for operation of the DDBMS.

Network Access Process

A Network Access Process (NAP) component is located at each computer in the network. It encompasses the communications hardware and software which links each computer to the rest of the distributed network.

The communication hardware and software area of the NAP handles functions such as message construction, message transmit timing, message acknowledgements, parity checking, checksum processing, and protocols. The NAP provides a "network description" to the NDBMS component. This description includes remote computer addresses and the paths linking them together. The status of all other computers in the distributed environment is also provided to the NDBMS. The NAP also fowards query data and command messages between the NDBMS and other remote NAPs.

Network Data Directory

The Network Data Directory (NDD) is a data component. It is used by the Network Data Base Management System to determine where a query should be processed. At this point a universal data model design feature must be presented. In the heterogenous environment each local data base has its own data model. Thus for an NDD to be applicable, each local data model must be transformed into a common descriptive model. This will be known as a universal data model. The specific entities which make up all the local directory elements will be transformed into the universal data entities. This information will be stored in the NDD. The task of translation will be performed by each NDBMS to create its own local NDD. When a query is presented to the NDBMS, the query's specific entities are translated to the universal data model by the NDBMS and located using the local NDD. The location or locations of the data are then returned to the NDBMS for further processing.

The existance of a universal data model is usually considered a "weak link" in the design of a DDBMS. This is because of the complexity of transforming local directory elements from different DBMSs into universal data entities. However, the need for such a model has been confirmed by many other researchers. At this time, no entirely general translation scheme has been accomplished. The development of a "specialized" universal data model will be a significant part of the NDBMS design. The long range goal of this development effort would be to eventually establish a general purpose translation scheme.

So far only the concept of a local NDD at each computer in the network has been presented. The design of this DDB system will allow for replicated data. Therefore, the proposed NDD scheme has two other considerations which will now be presented. They are a Centralized NDD (CNDD) and an Extended Centralized NDD (ECNDD). The CNDD and the ECNDD directory structures are characteristic of a replicated data base distribution strategy.

A Centralized NDD (CNDD) will be established at one of the computers. This directory will be a union of all the local NDDs. It will be in the same universal data model described above. The computer which maintains this directory will not need a separate copy of its own local NDD because the local NDD will be incorperated into the CNDD.

The purpose of the CNDD is to provide remote computers with remote directory information. The CNDD will maintain the data entity names and all locations where each data entity exists. Since replicated data will be allowed in this system design, a data entity may be located at more than one computer. It is the responsibility of the NDBMS at all computers in the network to ensure that the CNDD is accurate.

At this time the computer which maintains the CNDD has been identified as being a key element to the distributed system design. If it should fail, the ability to locate new remote data by another computer would be prevented. This problem presents the issue of how a CNDD gets established in the first place. In this design each "suitable" NDBMS will be provided with the capability of becoming the site which has the CNDD. The process involves polling each computer in the network and building the CNDD. The term "suitable" implies a computer which has a predetermined minimum storage capacity available for a CNDD. Should the computer which maintains the CNDD fail, the next "suitable" computer will become the CNDD site. This involves a short period of time when CNDD information is unavailable. Should all "suitable" computers be unavailable, then the network continues with the information is has at hand. In some instances, acknowledging that a query cannot be processed at the current time must be accepted. This brings up the second additional NDD consideration, that of an Extended Centralized NDD.

This design will establish an Extended Centralized NDD (ECNDD) capability at each computer. This should provide faster response times to queries and reduce overall communication costs. The tradeoffs for this capability will include the following four costs. First, the cost of storing the extended directory. *Second, the cost of storing* the address of copied data in the centralized directory. Third, the cost of storing a more complicated update procedure. Fourth, the communication cost of updating other ECNDDs.

The extended feature will be implemented in the same format as the local NDD. The ECNDD will be established at any computer when a remote data entity location is required. When a local NDBMS obtains the location(s) of required remote data from the CNDD, the information is appended to the local NDD and becomes the ECNDD. Do not be confused by actual replicated data and the directory information indicating where the replicated data is located. The time to replicate data and where is a function of the NDBMS, which is discussed later.

Since the ECNDD could conceivably grow to the size of the CNDD, a decision needs to be made for when to purge an extended item from the ECNDD. For resolving this decision the ECNDD will be limited to "N" items. The value for N should be determined experimentally. Once the ECNDD is filled, the next required listing in the ECNDD will replace an existing extended item. The replacement process will use a Least Recently Used (LRU) algorithm. A counter field will be added to each ECNDD item. Each counter field is initially zero. Every reference by the NDBMS to an ECNDD item will increment the associated counter. The item with the lowest counter field value is replaced. An alternative would be a timestamp purge of an item. Each time an ECNDD item is referenced, a timestamp would be attached to it. If the ECNDD is filled, then the item that is replaced would be the one with the longest time since last referenced. Both the LRU and timestamp algorithms use extra storage and require additional software. However, it is felt that the LRU algorithm is a simpler and more flexible procedure for resolving the replacement issue.

Another issue related to the NDD is the problem of which computers have copies in their NDDs of other computer's data. For example, suppose computer A replicated a part of computer B's data. Let's refer to this replicated data as a "Parts" inventory. Computer A must report this informationto the CNDD. Now computer C requests "Parts" location(s) from the CNDD. Computer C will be told that it is located at computers A and B. Lets assume that computer C's query is completed and that time elaspes. Now computer A decides to purge "Parts". Computer A not only deletes the "Parts" listing from its own local NDD but also informs the CNDD to delete the fact that computer A has a copy of "Parts". At this time the CNDD and computer C's ECNDD are not the same. This design will implement a "Do Nothing" policy. First, lets examine the alternative. The CNDD computer would have to record not only which computers have replicated data but also which computers have requested remote data information. This would quickly turn into a major bookkeeping problem. The "Do Nothing" policy requires only that a local NDBMS comfirm that a data entity still exists at a remote computer before continuing to process a query. A single message transmitted by the local NDBMS and positively acknowledged by the remote NDBMS is required. Data locations obtained directly from the CNDD need not be confirmed since the CNDD is always kept current.

One final issue related to NDDs is what happens when a computer goes down. In this case the responsibility of establishing its local NDD rests with its own NDBMS. All ECNDD listings are lost and must be rebuilt from the CNDD as needed.

Network Data Base Management System

The network data base management system (NDBMS) is the final network related component of the DDBMS. It has eight functions which are required because of the distributed environment. These eight functions are:

(1) User interface;

(2) Local DBMS interface;

(3) NAP interface;

(4) Local NDD interface;

(5) Query processing procedure;

  (5a) Universal data model translator;

  (5b) Repication process;

  (5c) ECNDD process;

  (5d) Universal query language translator;

(6) Concurrency control process;

(7) Backup process;

(8) CNDD process.

The function of the NDBMS is to provide an interface between the user and the data base. When the NDBMS processes a request for data, it accepts the query in the user query language of the local DBMS. The specific entities (the names of data items in the query) of the request are translated into universal data entities. These entities are compared with the information found in the NDD. If the request requires only local data then the original request is passed to the local DBMS for processing. No translation of the request is made except for those requiring NDD comparisions. The local DBMS processes the request and the result is returned back to the NDBMS. The NDBMS then provides the result to the user. The NDBMS is involved in controlling this processing for concurrency control reasons which will become evident later. If the request requires remote data, then special processing takes place.

To process remote data the NDBMS will have a universal query language translation capability. Although a general universal query language does not exist at this time, a specialized query language can be adapted which will use the NDD entities for reference. When the NDBMS requires remote data manipulation, a query is built which can be forwarded to the remote computer that has the necessary data. The query is then decomposed at the remote computer's NDBMS into its own query language for processing. The resulting data is then returned to the original computer's work area for further processing.

Up to this point, nothing has been mentioned about optimizing remote request strategies. This is an important function of the NDBMS. To accomplish this let's return to the discussion of the Network Access Process (NAP). The NAP provides the NDBMS with remote computer addresses and paths linking them together. The NDBMS obtains all the locations of remote data from the NDD and then using the NAP data computes the optimum processing site(s). The processing of requests requiring remote data is considered for two types of requests, simple and compound requests. A simple remote request is one for which all the data is located at one remote computer. In this case the NDBMS determines to which computer to forward the request for processing. The request is processed completely at that remote computer and the result returned. A compound request involves decomposing the request into a set of local and remote requests or multiple remote requests. In this case the query processing strategy developed by Eugene Wong will be used by the NDBMS for selecting the processing method. Recall this is a recursive technique which finds an optimum processing strategy by minimizing data movement.

The query processing strategy also will maintain a table to indicate which remote data entities are used and how often. The purpose of this function is to determine if and when to replicate remote data based on usage. The table referenced above will be physically located in the ECNDD. This uses the fact that if remote data is being considered then that particuliar computer has a nonempty ECNDD. The ECNDD described earlier identified a counter field will be added to each ECNDD item. Each counter field is initially zero. Every reference by the NDBMS to an ECNDD item will increment the associated counter. If the counter reaches "M" (an experimentally determined value) before the item is purged from the ECNDD then the data for this item is replicated at the local computer if possible. This process includes three maintenance functions. First, removing the item from the ECNDD. Second, adding the item to the local NDD. Third, informing the CNDD of the replication. The automated function of replication of remote data into a local data base is extremely complicated. However, with the establishment of a universal data model and translation capability, the replication function should be obtainable.

Another function of the NDBMS is concurrency control. When replicated data is allowed, as in this design, the monitoring of updates is important. The majority voting algorithm will be used for this purpose. This algorithm is selected for two reasons. First, and most importantly, it has been shown to work correctly. That is, all copies of the replicated data converge to the same value and internal data base consistency is preserved. Second, the algorithm is deadlock free.

The NDBMS will provide a backup and initialization function. When a computer fails, all other computers in the network are informed by their associated NAPs. When the computer which maintains the CNDD is informed, it checks its CNDD for replicated data entities which were involved with the lost computer. If a replicated entity is found then a message is transmitted to the remote computer involved. The message commands the remote computer to begin maintaining an audit trail for the replicated data. This audit trail will be maintained for 24 hours (arbitrarily choosen) or until a predetermined number of enteries in the audit trail are made. This cutoff mechanism is used to decide whether implementing an audit trail recovery is more cost beneficial than a replication effort of the data.

At the time the down computer comes active again the local DBMS will recover its own local data base. When this is completed the NDBMS creates the local NDD. The local NDD is then compared to the CNDD to locate any replicated data. The decision is then made by the recovering computer's NDBMS as to whether it will update using the other computer's audit trail or replicating the data.

Physical Network Data Directory Structure

This section provides for the design of the three data structures that will be used by NDBMS directory software processes. The structures, described in order of complexity, are the LNDD, CNDD, AND ECNDD.

The LNDD will be a structure which contains all universal data model entity names that are stored in the local data base. Its size will be determined by the number of entities stored in the local data base. Let's assume the local data base has "k" entity names. It must be searchable in the shortest possible time since it will be referenced by every query submitted to the NDBMS. Additionally, it must be able to handle additions or deletions routinely. The solution presented here is based on the binary AVL tree data structure. The nice feature of an AVL tree is that the tree structure remains balanced with respect to the heights of subtrees. The AVL definition also requires that all subtrees be height balanced. As a result, dynamic retrievals can be performed in O(log k) worst case time, where "k" is the number of local data base entity names. In the case of a binary search tree, a possible alternative, the worst case time is O(k). Additionally, a new node in the tree can be entered or deleted in O(log k) worst case time. Its declarative structure in Pascal is:

```
TYPE
  LNDD = ↑LNDDRECORD;
  LNDDRECORD = RECORD
                 ENTITYNAME : ALFA;
                 HEIGHTDIFFERENCE : INTEGER;
                 LEFTCHILD : LNDD;
                 RIGHTCHILD : LNDD
               END;
VAR
  LNDDTREE : LNDD.
```

The Centralized Network Data Directory (CNDD) will be a structure which contains all entity names used throughout the entire distributed data base. However, the CNDD must also provide the locations of all replicated data when required. For speed reasons, the CNDD will also be a binary AVL tree construct similiar to the LNDD structure. An additional secondary linked list record is added to the declarative structure for indicating replicated data locations. The declarative structure in Pascal is:

```
    TYPE
      LOCATION = ↑LOCATIONRECORD;
      LOCATIONRECORD = RECORD
                           COMPUTERADDRESS : INTEGER;
                           NEXTADDRESS : LOCATION
                         END;
      CNDD = ↑CNDDRECORD;
      CNDDRECORD = RECORD
                       ENTITYNAME : ALFA;
                       REPLICATIONS : LOCATION;
                       HEIGHTDIFFERENCE : INTEGER;
                       LEFTCHILD : CNDD;
                       RIGHTCHILD : CNDD
                     END;
    VAR
      CNDDTREE : CNDD.
```

The Extended Centralized Network Data Directory (ECNDD) will be a structure which contains remote data locations for the local computer's NDBMS. It's length will be considerably shorter than that of the LNDD. It will be consulted after failing to locate an entity name in the LNDD. Since it is relatively short, it will be a linked list with no sorted pattern. It consists of an entity name, a counter and a secondary linked list. The counter is used for determining when to replicate data. The secondary linked list is used for indicating replicated data locations just as in the CNDD structure. The declarative structure in Pascal is:

```
TYPE
  ECNDD = ↑ECNDDRECORD;
  ECNDDRECORD = RECORD
                  ENTITYNAME : ALFA;
                  REPLICATIONS : LOCATION;
                  COUNTER : INTEGER;
                  NEXTENTITY : ECNDD
                END;
VAR
  ECNDDLIST : ECNDD.
```

Conclusion

DDBMSs are often only talked about and rarely implemented. This research was an attempt at designing a DDBMS that could be implemented in the AFIT DEL. The top level design presented is a combination of good ideas that have been uncovered during the research. Since only the NDD component was developed in the lower level design stage, there is much flexibility for alteration of the rest of the design. As time progresses some of the solutions to questions left unanswered will undoubtably surface in DDB literature. It is hoped that this research will contribute in some way to a more advanced understanding of distributed data base management systems.

Vita

Eric F. Imker was born on August 30, 1955 at Riverside, California. He graduated from Mascoutah Community High School at Mascoutah, Illionis in January, 1973. He attended San Jose State University in California from which he received a Bachelor of Science degree in General Engineering (option Computer Science) in May 1977. While at San Jose State he joined the USAF ROTC program and received a reserve commission upon graduation. He went on active duty in July 1977 and was assigned to Headquarters Tactical Air Command, Langley Air Force Base, Virginia where he spent four years in the Office of Data Automation. In June 1981 he entered the School of Engineering at the Air Force Institute of Technology. While at AFIT he was commissioned a regular officer.

Permanent address: 2395 Delaware Ave #97
Santa Cruz, CA 95060

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFIT/GCS/EE/82D-21	2. GOVT ACCESSION NO. A124 821	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN OF A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM FOR USE IN THE AFIT DIGITAL ENGINEERING LABORATORY		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Eric F. Imker Capt USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE DECEMBER 1982
		13. NUMBER OF PAGES 123
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Reproduced from best available copy.

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

4 JA

Approved for public release: IAW AFR 190-1
[illegible]
LYNN E. WOLAVER
Dean for Re[illegible] and Professional Developme
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Data Base Management System
Network Data Base Management System
Distributed Directory System
Data Base Directory
Network Data Directory
Distributed Data Directory

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A distributed data base management system (DDBMS) was designed with the goal of providing the AFIT School of Engineering a research tool. The objective was to use state-of-the-art knowledge in a design that would provide an experimental testbed to further advance DDBMS knowledge.

Toward this goal, an extensive investigation was made into distributed data base systems. Numerous alternatives were presented in the areas of

configurations and architecture, data allocation, directory management, query processing, concurrency control, reliability, integrity and security. This background discussion includes advantages and disadvantages of the alternatives.

A top level system design was presented which includes replication of data, a universal data model and query language, a centralized and extended centralized directory system and the majority voting concurrency control algorithm. Due to the complexity and size of the development effort, a detailed design of only the directory system was made in this research effort. Follow-on development efforts over the next few years will be required to complete this research project.

END